# Machine Learning to Balance Web Applications using SDN

**Prakruthi M Krishna[1], Sanjana S Rao[2], Simran Suresh Bhurat[3], Tata Venkatesh[4], Prof. Vani K A[5]**

[1,2,3,4]Students, [5]Assistant Professor
Information Science and Engineering, Dayananda Sagar College of Engineering

***Abstract*: Using the consistent hash functions, core of an existing load balancing mechanism will be static, which is actually expected to be able to map both the servers and client workloads to a hash circle, uniformly. But choosing a Load Balancing algorithm statically does not work in many situations when the applications have unknown behavior or dynamic one which is very common in this world. Traditional networks use hardware components which are expensive, complicated to deploy and require human intervention to work consistently. These limitations make load balancing among servers, complex, expensive and non-scalable. Based on the principles of SDN, we tackle these challenges using an approach that checks and also collects the information about the application requests which are incoming dynamically, and finally according to this analyzed data the model makes a decision of selecting the most suitable Load Balancing algorithm.**

***Keywords*: Load Balancing Algorithm, Dynamic Load Balancer**

## Introduction

In today's world with a faster growing rate of latest technologies and trends on the internet, multiple users access a portal or a website simultaneously. Online businesses and retailers must ensure that they are able to accommodate a vast number of customers at the same time on their site in order to decrease delays and avoid server crashes. Thus, it is essential to incorporate the concept of load balancing in all the online websites.

In distributed processing architectures, it is optimum to achieve load balancing which is of significant importance while we struggle with network overhead issues. A well-balanced load in the network supports to optimize and utilize the available resources in a way where we can maximize the throughput, minimize the response time, and avoid resources to be overloaded in the network. For the purposes of reducing the heavy-traffic network flux and lessening the risk of one server becoming the main overhead contributor, multiple data-centers adopt hardware resources particularly dedicated to achieve load balancing. This also ensures supporting a large number of users. Yet, various technical complications and the increasing costs in the deployment of these hardware systems present a major challenge.

Load balancing offers various advantages, few of which are scalability, manageability, availability and security of webpages. Load balancing helps by distributing the load across multiple servers which indeed ameliorates the problems connected with the scalability of the applications as well as server cluster. Redirecting the traffic to different servers if there‟s an application failure or server crash, is one by the load balancer. It permits network and server administrators to move an application from one server to another improving the manageability.

**There are 2 types of load balancing algorithms:**
Static load balancing algorithms are formed for systems with fewer variations in load. Here, the entire traffic is equally distributed among the servers. This algorithm entails deep information of server resources in order to ensure good performance of the processors, determined at the initial stage of implementation. Nevertheless, load shifting does not depend on the present state of the system. One of the major drawbacks of static load balancing systems is that they do not utilize node performance data to make decisions while distributing the load. Round robin algorithm is one example for a static load balancer.

Dynamic load balancing algorithms begins by searching the lightest server in the overall network. The server with least load is offered preference for load balancing. This algorithm acquires real time communication with the network which will enable in improvising the traffic of the system. The present state of the system is used to achieve the load balancing. The forte of dynamic load balancer is to make load transferring choices to the definite current state. In such a system, processes in real time, move from an extremely utilized machine from an underutilized machine. Least connections algorithm is an example for dynamic load balancing algorithm.

**Types of Load Balancers**

- **Round Robin:** Here, the requests are server among a group of multiple servers in a consecutive manner.
- **Least Bandwidth:** This algorithm determines the traffic of each and every server in megabits per second (Mbps), and the client requests will be served by the server with the minimum bandwidth, i.e., Mbps of traffic.
- **Least Time:** Requests are served by the server with selected by a formulation combining the fastest response time and least active connections.
- **Hash:** Here a key called hash is used. The algorithm distributes requests depending on a key defined by the used which could be the request URL or the client IP address.

- **Least Connections:** The server with the fewest current connections is determined and the request is forwarded to that particular server. The relative computing capacity of every server is considered.

**Software Defined Networks:** Software Defined Networks (SDN) is a computer network that enables a smooth and convenient network that enables network flow control method. SDN Mechanism is the technology which separates the underlying data plane that forwards network traffic from the control plane management of network devices. It needs very low investment and helps to lessen costs, as well as huge number of users are being provided with the benefits.

SDN is a software implementation of switches for transport of data. It creates a flow table for lookup operation as soon as a data packet reaches the server. For every network flow, when there are flow changes the headers and counters will be updated and actions will be imposed. The header information will be recorded into a database which will then be used by an OpenFlow switch that will process the data flow in accordance with the header records. SDN addresses the fact of static architecture of traditional networks which is decentralized and complex, whereas the present world networks require greater flexibility and earlier troubleshooting. SDN tries to centralize the network intelligence in a single network component by disassociating and decoupling the data plane that is the forwarding process of network packets from the control plane that is the routing process. The control plane is the brain component of the software defined architecture which has one or more controllers. It is where the entire intelligence is combined.

The SDN architecture has 3 layers. Application layer is the first layer and it maintains the system applications or the business. Control layer is the second layer which has all the software that controls the SDN software and the network services which maintains all the sessions end devices. Infrastructure layer is the third layer or the last layer which has all the network devices that participates in the network routing process. The speed of the network is dependent on the data flow and can be controlled in the infrastructure layer and the control layer.

**Machine Learning:** Machine Learning involves a study that enables computers the capacity to learn by experiences without actually being programmed. This has been one of the most exciting and interesting technologies that we have developed. This allows machines to learn and build intelligence just like human beings. The process of acquiring knowledge starts with observations or data, which includes examples, instructions, direct experiences, or patterns in data and make improved decisions in the future considering the examples that are fed. The main goal here is allowing the computers to learn by themselves without the intervention of humans or assistance and consequently fine-tune the actions.

Supervised machine learning algorithms apply previously learnt knowledge to the incoming data with the help of labeled examples and predict the future events. It begins with the analysis of a recognized training dataset; the learning algorithm then produces an inferred function that makes predictions regarding the output values. The system offers targets for every new input after adequate training. The learning algorithm also compares its output with the accurate, intended output to discover errors and amends the model accordingly. Unsupervised machine learning algorithms, in contrast are made use of when the information that is utilized to train is neither labeled nor classified.

Unsupervised learning discovers how the systems can infer a function to describe a hidden structure from non-labeled data. Here the system discovers the data and draws inferences from large collections of data to describe hidden structures from unlabeled data, rather than figuring out the right output. Machine learning allows analysis of enormous quantities of data. Even though produces faster, precise results to identify gainful opportunities or hazardous risks, it involves extra time and resources to train it accurately. Combining machine learning with AI and cognitive technologies makes it more enhancing in processing of huge volumes of data and information.

The machine learning we have used is Logistic Regression which is a supervised learning model, to get trained and predict the optimal type of the request.

## Literature Survey

The traditional LB mechanisms are categorized into mainly four major types [1] based on the client side, middle layer, based on DNS, and the transport layer. In the load balancing mechanism based on the client side, the clients primarily collect each running parameters of the server from the server clusters either non-periodically or periodically, and to achieve load balancing, it forwards that particular request to different servers.

Next is the most general method of LB based on the middle layer which in turn requires the server called the reverse proxy server. To balance the load in the server clusters, this reverse proxy server requests the back-end servers, also forwards the cached data back to the clients. The acceleration mode accelerates the access speed of the static pages. To improve the performance, the reverse proxy server combines the load balancing technology with the caching technology. However, by developing a reverse proxy for each service is a substantial demand.

LB mechanism based on DNS is smooth but it has several issues. Because, DNS server does not know about the differences among the servers, and hence cannot show the present state of the servers. But it can actually send an access to the non working servers when there is an increasing growth of the server load on currently using servers. The load balancing scheme based on transport layer sends the client‟s requests directly to the load balancing server. The load balancing server will then send the requests to the back-end servers according to policies such as LVS [3], VS/NAT, etc. Although this approach balances the server load, it often demands additional hardware resources and hus proven costly. Due to such hardware limitations, the load balancing among the server clusters is not only complex but also expensive, and is susceptible of poor scalability.

The emergence of SDN architecture facilitates effective strategies to counteract such load balancing issues. 2.2. The SDN architecture encompasses a decoupling layered architecture which divides the data level access from the control level access [4]. The SDN process architecture can be categorized into a three-layer system structure [5] including application, control and data. Rather than a simple extension of the traditional network architecture, this three-layer structure of SDN is a disruptive innovation [6].

The control level has the network operating system and applications, while the data level incorporates standard protocol supports for the network hardware equipment. The centralized network control of SDN is effective in resolving the susceptible issues of the traditional network devices. Moreover, SDN supports independent programming of the network control system in the management mode, and instantaneous upgrading of which can be achieved through the network application interface. The application layer of the SDN architecture provides users with a rich variety of API interfaces, which can be used to develop our own development module with desired functionalities [7] based on individual business needs. OpenFlow, one of the SDN mainstream southbound interface protocol [8], is one of the fundamental elements required for building SDN solutions.

OpenFlow is the communication protocol defined between the control layer and the infrastructure layer in SDN architecture [9,10]. Switches are responsible for applying appropriate actions on the arriving packets and update every action on their flow table entry. Using such entries in the flow tables, switches simple forward packets without considering to construct or modify the flow tables. The SDN controller will create and install a rule in the flow tables for the necessary packets. Also, the SDN controller can manage the flow tables of all the switches simultaneously. OpenFlow uses the concept of flows to identify network traffic based on matching rules that can either be statically or dynamically programmed by the SDN control software. OpenFlow-based SDN architectures provide extremely granular control privileges, by the way of enabling the network to react to real-time changes of both the application and the service users [11]. OpenFlow-based SDN technologies can improve the network bandwidth capabilities, can handle the dynamicity of the applications and can significantly lessen the operation and management complexity [12].

The authors made use of SDN (Software Defined Networks)[13] as a smooth and cost-effective network flow control as against the hardware resources used in traditional networks. They came up with a new method to achieve effective load balancing among server clusters using the SDN controller which measures the real-time server response time. They included the floodlight controller module and Openflow environment to choose the server with minimum or most stable response time. The server with minimum response time implies maximum load capability and hence maximum performance. They proposed a new LB algorithm [14] that collects information periodically on the traffic through the server response time and switch ports. Then this load balancer adapts the strategies for web requests‟ allocation. Also, a comparison between round- robin, random selection, and LB by Server Response Time approaches in the SDN environment was proposed. This method achieves greater throughput and least response time when compared to the other LB methods by considering the traffic through server response times and the switch ports simultaneously.

[15] Based on variance analysis, they presented a probabilistic method of load balancing which was used to manage the traffic flows dynamically to support huge mobile users in SDN networks. They provided probability-based selection algorithm to solve the complicated selection problem that network faces and to redirect traffic dynamically with the OpenFlow technology as well. When compared to the existing LB methods, the method proposed by them was able to solve the observed deficiencies of the old traditional methods such as greater cost, least reliability, and very low extensibility. By using the probability-based selection algorithm which is integrated with the SDN controller, the demand from massive mobile users could be satisfied at run-time.

[16] The authors have considered two major problems – Load oscillation problem and time management problem and based on these, they have come up with a solution. Here, the authors have adopted logically-centralized control plane as multiple-controllers architecture. So, in the proposed architecture, there are multiple local controllers and a super local controller. For load measurement, to denote the load of the controller, they use the total arrival rate of PACKET_IN events and the sending rate of PACKET_IN events would represent the load of switch. The switches selection algorithm selects the furthest switches group with the overloaded controller to migrate when there are multiple groups and also reduces latency.

**Proposed System**

To create a learning-based Load Balancer that can emerge as a solution to the varying and also unpredictable behavior of the application that enables the performance and also the scalability in areas like IoT and AI Data Processing. The project is based on developing an efficient load balancing scheme using the SDN architecture. By making use of ML, we present an approach to automate the LB mechanism and algorithm decision at run time. Our system considers the various features that make up the context where the application is being run. And according to the set of features collected, this gives us the ability to select various heuristics for the execution of the same application during run time and decide where to direct the request. Hence, we want to create a Learning based Load Balancer that enables the scalability and performance in areas like IoT, AI, Data Processing, etc.

**System Architecture**

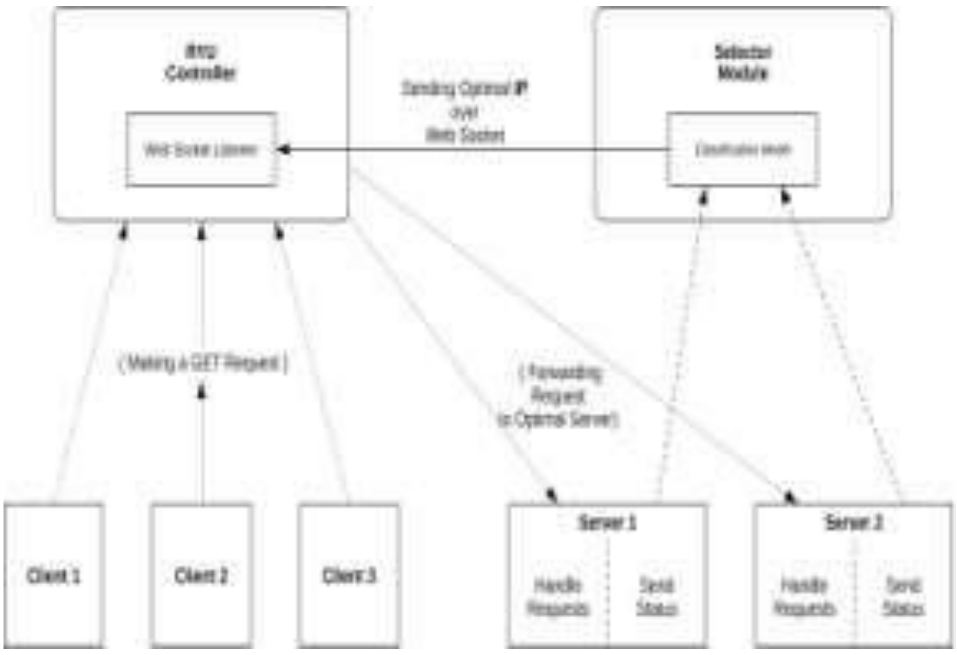

Fig 1.0

The Fig 1.0 represents the entire architecture of our load-balancing system and how the different modules within it communicate with each other. The various components are: RYU Controller, Selector Module, Client and the Server

**1) RYU Controller:** RYU is a component-based software defined networking framework that provides software components which we require for creating and managing the network topology, and its various protocols for managing network devices include OpenFlow it was natural for us to choose this framework. It gives us the capability to create web servers corresponding to WSGI. We used this function extensively to connect the Selector Module with the Controller within the system. The Controller is the module responsible for receiving and forwarding requests between clients and servers within the network that the controller is responsible for. The controller is wired with a socket listener that will be listening for a JSON object that contains information about which servers are best optimal for a given type of request. Once the controller receives the IP from the Selector Module, the controller when receives the information of optimal IP to be used, the accordingly forwards the request to the optimal server in the network by the controller. Allowing us to manage network gear in our way and configure the gear by using OpenFlow protocol.

**2) Selector Module:** The Selector Module is where we integrate our trained Machine Learning Model [4] which will take the Trip-Time, Ram Usage and CPU Usage (features for the model) as input and output the best type of request the server with those given stats could handle. The machine learning model is trained on 200 synthetically created data points where each data point is a set containing the Trip-Time, Ram Usage and CPU Usage and label that is either a 1 or 0 corresponding to either Media or Test type of request. The machine learning model uses logistic regression to get trained and predict the optimal type of request to handle when give the features described above.

The function used in the ML model is Sigmoid, the Fig 1.1 is the representation of it.

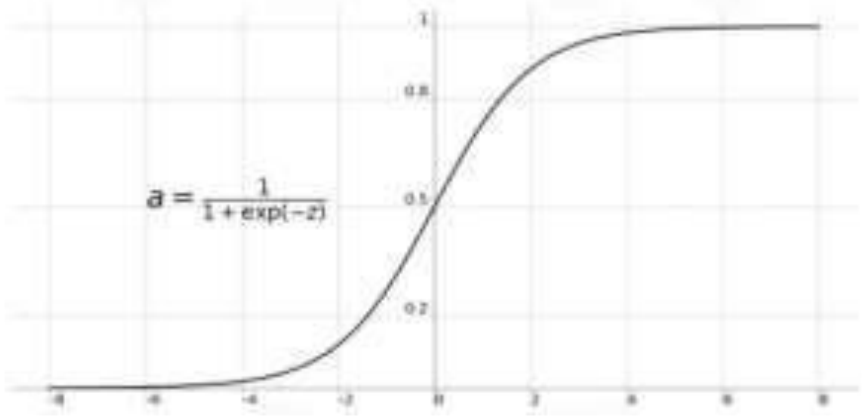

Fig 1.1

The pseudocode for our Machine Learning algorithm for classifying optimal request type for give features is as follows:

```
(1) initialize $\beta_{kj} \leftarrow 0, \Delta_{kj} \leftarrow 1$ for $j = 1, \ldots, d$, $k = 1, \ldots, K$
    for $t = 1, 2, \ldots$ until convergence
      for $j = 1, \ldots, d$
        for $k = 1, \ldots, K$
(2)       compute tentative step $\Delta v_{kj}$
(3)       $\Delta\beta_{kj} \leftarrow \min(\max(\Delta v_{kj}, -\Delta_{kj}), \Delta_{kj})$  (reduce the step to the interval)
(4)       $\beta_{kj} \leftarrow \beta_{kj} + \Delta\beta_{kj}$  (make the step)
(5)       $\Delta_{kj} \leftarrow \max(2|\Delta\beta_{kj}|, \Delta_{kj}/2)$  (update the interval)
        end
      end
    end
```

**3) Client:** The client in the system and in the Fig 1.0 refers to the host machine within the network that is requesting data which could be of type Media or Text. The response to the client will be sent according to the server that is best suited for the sent type of request.

**4) Server:** Server is the host within the network that has to process the incoming request from the client. Two processes run in parallel in every server within the network. One process is responsible for handling the incoming request from the client that has been forwarded by the RYU Controller, and the second process involves sending the information of itself i.e., the trip-time, ram usage and CPU usage to the Selector Module in equal intervals of time so the Selector Module can run the ML model with the received stats as input and decide which IP is best for which type of request.

**Experiments and Results**

**Experiment Setup:** This test requires a network topology consisting of 5 hosts connected to a single switch within a network. Of the five hosts, 3 hosts are clients that are sending random Test of Media type or text requests to the Controller IP and the controller will send the request to the appropriate server within the given time period. Mininet is a network emulator or perhaps more precisely a network simulation system. It also runs a host of end-to-end hosting, switches, routers and links to a single Linux kernel. It uses simple visuals to make a single application look like a complete network, using the same kernel, program, and user code. Mininet keeper behaves like a real machine that allows us to SSH into it and run arbitrary programs. Programs can send packets with what looks like a real Ethernet interface, with a given connection speed and delay. Packages are processed by what looks like a real Ethernet switch, router, or a middlebox, with a given number of queues. When two systems, such as the iperf client and server, are connected via Mininet, the rated performance should be the same as for two (slow) native machines.

**Environmental Limitations:** Mininet uses a single Linux kernel for all the virtual hosts; this means we cannot use BSD-based software, Windows, or another operating-system kernel. (Or we can attach VMs to the Mininet.). Mininet will not write us an OpenFlow controller; if we need a custom route or change behaviour, then we have developed a controller with the features we need. By default, the Mininet network is disconnected from the LAN and internet. However, we can use the NAT object and/or option --nat to connect your Mininet network to the LAN via Network Address Translation. It can also install a real (or virtual) hardware interface on your Mininet network. By default, all Mininet browsers share the Host file system and PID space; this means we have to be careful when using daemons that need to be stopped in / etc, and we need to be careful that we do not kill wrong practices by mistake. Unlike the simulator, Mininet does not have a solid view of real-time; this means that time ratings will be based on real-time, and that results faster than real-time (e.g. 100 Gbps networks) cannot be easily replicated.

The following command was used to create the topology sudo mn --topo single,5 --mac --controller remote --switch ovsk
Once the topology has been created, the wsgi based RYU-Controller and the Selector Module are started. The arguments in the above command are

a) topo: a topology object (not a constructor or template - the actual object!)
b) host: a constructor used to create Host elements in the topology
c) switch: a constructor used to create Switch elements in the topology
d) controller: a constructor used to create Controller elements in the topology

**Results:** Our Logistic Regression Model has been trained on 100 data points out of which 25 points have been used as the test set. The below matrix represents the confusion matrix of the Model.

$$\begin{bmatrix} 10 & 2 \\ 1 & 12 \end{bmatrix}$$

We can see the true positives, true negatives, false positives and false negatives from respective indices from the above matrix. Our ML model has shown an accuracy of 88% in choosing the most optimal server for a given type of request.

Below are the trained model plots:

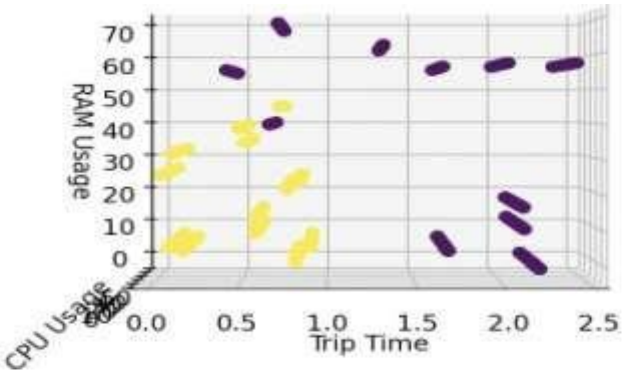

Fig 2.0

Each point in Fig 2.0 represents a server that is classified to be optimal of handling either a Media request or a Text request. Yellow represents Media Optimal and the Blue represents Text Optimal. And the above figure shows the model performance again Trip Time as one of the input features.

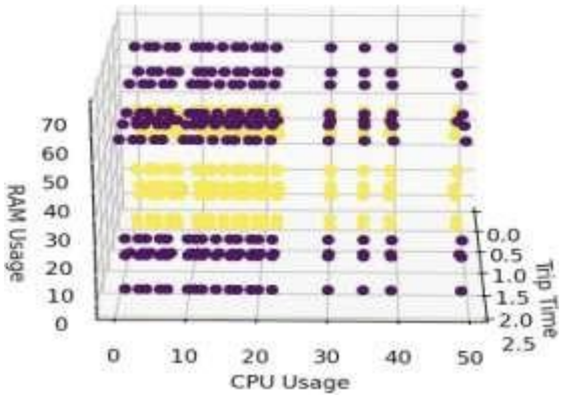

Fig 2.1

The Fig 2.1 represents the model performance against the CPU usage as the input feature and how the model classified various servers with different characteristics.

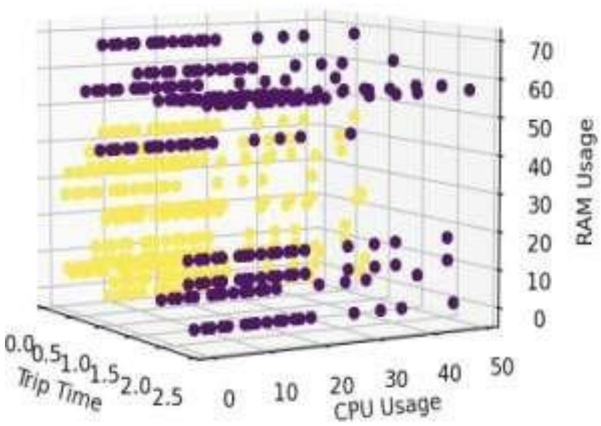

Fig 2.2

The Fig 2.2 represents the model performance against the RAM usage as the input feature and how the model classified various servers with different characteristics. And you can even notice all the independent features given to the model and the classification performed on the servers with respect to color.

## Conclusion

The hardest problem is load balancing in parallel machines which can result in huge impacts on the application performance and it is a very thoroughly studied subject. In order to solve the issue in a timely manner, dynamic load balancers used procedural solutions that are prevalent in the literature. Imbalance of load is a complicated issue which hampers scalability and performance of various parallel and diverse applications. Here, the important task is to select the best LB algorithm suitable for applications that are not predictable, dynamic, irregular. Here, by using ML, we have presented an approach for that automates the decision of an LB algorithm during the run time. Also, we showed that, in accordance with the given inputs, the application is presented at a particular instant of time, the load balancer is designed to switch between the traditional load balancers, leading to a better performance.

## Future Work

The Machine Learning model which we have used here takes the Trip-Time, Ram Usage and CPU Usage as inputs. We want to improvise our model by adding more features and by considering a greater number of data points (type of requests). We intend to create the dynamic Load Balancer which takes the input and switch between the traditional LB and to extend the present solution to other Load Balancing Algorithms. Lastly, we want to use this ML-based approach to other application programming libraries as future work.

## References

[1] Z. Shang, W. Chen, Q. Ma, et al., Design and implementation of server cluster dynamic load balancing based on OpenFlow, 2013 International Joint Conference on Awareness Science and Technology and Ubi-Media Computing, (iCAST-UMEDIA), IEEE, 2013, pp. 691–697

[2] Z. Xu, R. Huang, L. N. Bhuyan, Load balancing of DNS-based distributed web server systems with page caching, Tenth International Conference on Parallel and Distributed Systems (ICPADS), 2004, IEEE, pp. 587–594

[3] R. Tong, X. Zhu, A load balancing strategy based on the combination of static and dynamic, 2nd International Workshop on Database Technology and Applications (DBTA), IEEE, 2010, pp. 1–4

[4] S. Yu, IEEE approves new IEEE 802.1aq shortest path bridging standard [EB/OL]. [2012-05-08] https://www.techpowerup.com/165594/ieee-approves-new-ieee-802-1aq-shortest-path-bridging-standard

[5] M. K. Shin, K. H. Nam, H. J. Kim, Software-defined networking (SDN): A reference architecture and open APIs, International Conference on ICT Convergence (ICTC), IEEE, 2012, pp. 360–361

[6] A. Malishevskiy, D. Gurkan, L. Dane, et al., OpenFlow-Based Network Management with Visualization of Managed Elements, Research and Educational Experiment Workshop (GREE), 2014 Third GENI, IEEE, 2014, pp. 73– 74

[7] S. Huang, J. Griffioen, K. L. Calvert, Network Hypervisors: Enhancing SDN Infrastructure, Comput. Commun., 46, 2014, 87–96

[8] A. Lara, A. Kolasani, B. Ramamurthy, Network innovation using OpenFlow: A survey, IEEE Commun. Surv. Tutor., 16 (1), 2014, 493–512

[9] N. McKeown, T. Anderson, H. Balakrishnan, et al., OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Comput. Commun., Rev. 38 (2), 2008, 69–74

[10] C. Rotsos, N. Sarrar, S. Uhlig, et al., OFLOPS: An open framework for OpenFlow switch evaluation, Passive and Active Measurement, Springer, Berlin, Heidelberg, 2012, pp. 85–95

[11] M. Kobayashi, S. Seetharaman, G. Parulkar, et al., Maturing of OpenFlow and Software-defined Networking through deployments, Comput. Netw., 61, 2014, 151–175

[12] H. Yin, T. Zou, H. Xie, Defining data flow paths in software-defined networks with application-layer traffic optimization: U.S. Patent Application 13/915,410[P]. 2013– 6-11

[13] Hong Zhong, Yaming Fang, Jie Cui, Survey on LBBSRT: An efficient SDN load balancing scheme based on Server response time, 2017

[14] Mahmood Ahmadi, Kiarash Soleimanzadeh, Survey on SD-WLB: An SDN-aided mechanism for Web Load Balancing, 2018

[15] Hong Zhong, Qunfeng Lin, Jie Cui, Runhua Shi, and Lu Liu, Survey on SDN-LB:An Efficient SDN Load Balancing Scheme for Massive Mobile Users, 2015

[16] Yaning Zhou, Ying Wang, Jinke Yu, Junhua Ba, Shilei Zhang, Survey on Load Balancing for Multiple Controllers in SDN Based on Switches Group, 2017