

Article

Enhancing QoS of Telecom Networks through Server Load Management in Software-Defined Networking (SDN)

Khawaja Tahir Mehmood ^{1,*}, Shahid Atiq ¹ and Muhammad Majid Hussain ^{2,*} 

¹ Department of Electrical Engineering, Khwaja Fareed University of Engineering & Information Technology, Rahim Yar Khan 64200, Pakistan; shahid.atiq@kfueit.edu.pk

² School of Engineering and Physical Sciences, Heriot-Watt University, Edinburgh EH14 4AS, UK

* Correspondence: ktahir.kfueit1001@gmail.com (K.T.M.); muhammad.hussain@hw.ac.uk (M.M.H.)

Abstract: In the modern era, with the emergence of the Internet of Things (IoT), big data applications, cloud computing, and the ever-increasing demand for high-speed internet with the aid of upgraded telecom network resources, users now require virtualization of the network for smart handling of modern-day challenges to obtain better services (in terms of security, reliability, scalability, etc.). These requirements can be fulfilled by using software-defined networking (SDN). This research article emphasizes one of the major aspects of the practical implementation of SDN to enhance the QoS of a virtual network through the load management of network servers. In an SDN-based network, several servers are available to fulfill users' hypertext transfer protocol (HTTP) requests to ensure dynamic routing under the influence of the SDN controller. However, if the number of requests is directed to a specific server, the controller is bound to follow the user-programmed instructions, and the load on that server is increased, which results in (a) an increase in end-to-end user delay, (b) a decrease in the data transfer rate, and (c) a decrease in the available bandwidth of the targeted server. All of the above-mentioned factors will result in the degradation of network QoS. With the implementation of the proposed algorithm, dynamic active sensing server load management (DASLM), on the SDN controller, the load on the server is shared based on QoS control parameters (throughput, response time, round trip time, etc.). The overall delay is reduced, and the bandwidth utilization along with throughput is also increased.

Keywords: DASLM; data transfer rate; end-to-end user delay; HTTP; maximum available bandwidth; QoS; server load management; SDN



Citation: Mehmood, K.T.; Atiq, S.; Hussain, M.M. Enhancing QoS of Telecom Networks through Server Load Management in Software-Defined Networking (SDN). *Sensors* **2023**, *23*, 9324. <https://doi.org/10.3390/s23239324>

Academic Editors: Wilfried Gappmair, Zahir M. Hussain, Erich Leitgeb, Maja Matijašević and Mario Kusek

Received: 25 September 2023
Revised: 15 November 2023
Accepted: 17 November 2023
Published: 22 November 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Owing to recent advancements in cloud computing, big data applications, and complex network architecture with machine learning applications, legacy networks cannot fulfill the demands of system administrators and network users. Currently, the primary goal of every network designer is to provide fast and reliable data transformers (wired or wireless). The other important aspect is maintaining QoS, that is, achieving better throughput, greater bandwidth, and less latency delay. Virtualization reduces the number of physical components in the network, which can result in energy-efficient and maintenance-free systems. A network system that provides all of the applications mentioned above is a Software-Defined Networking (SDN). At the same time, SDN is a centralized-based [1] virtual control system that provides an entire view of the network underlying its control layer. SDN reduces the extra burden by separating the data and control layers from the network components [2] and performs a virtual control process using an SDN controller based on the logical instructions provided by the application layer. A research article [3] showed the performance-based results of a heavily loaded network environment consisting of 256 servers for data modulation by testing different network systems. They concluded that SDN performance in data modulation was 47% greater than the legacy network. In

Hadoop applications [4], SDN outperforms legacy networks regarding controllability, scalability, and flexibility in normal forwarding modes. SDN with a centralized controlled technique can provide better controllability, scalability, and reliability than traditional physical networks. Currently, all networking architectures employ software (IP—version: 6, BGP—version: 4, MPLS—version: 3, VMware—version: 17.5.0, etc.) for data management and transformation [5]. All protocols mentioned above can be easily manipulated in an SDN virtual environment; with this flexibility, SDN is becoming a vital networking force [6,7]. The QoS parameters are the primary deterministic performance tools in tele-traffic engineering. The QoS results of SDN with separate data and control layers [8,9] are far superior to legacy networks. Figure 1 shows the layered structure of an SDN. As previously mentioned, the SDN consists of three layers, namely, the application, control, and infrastructural layers.

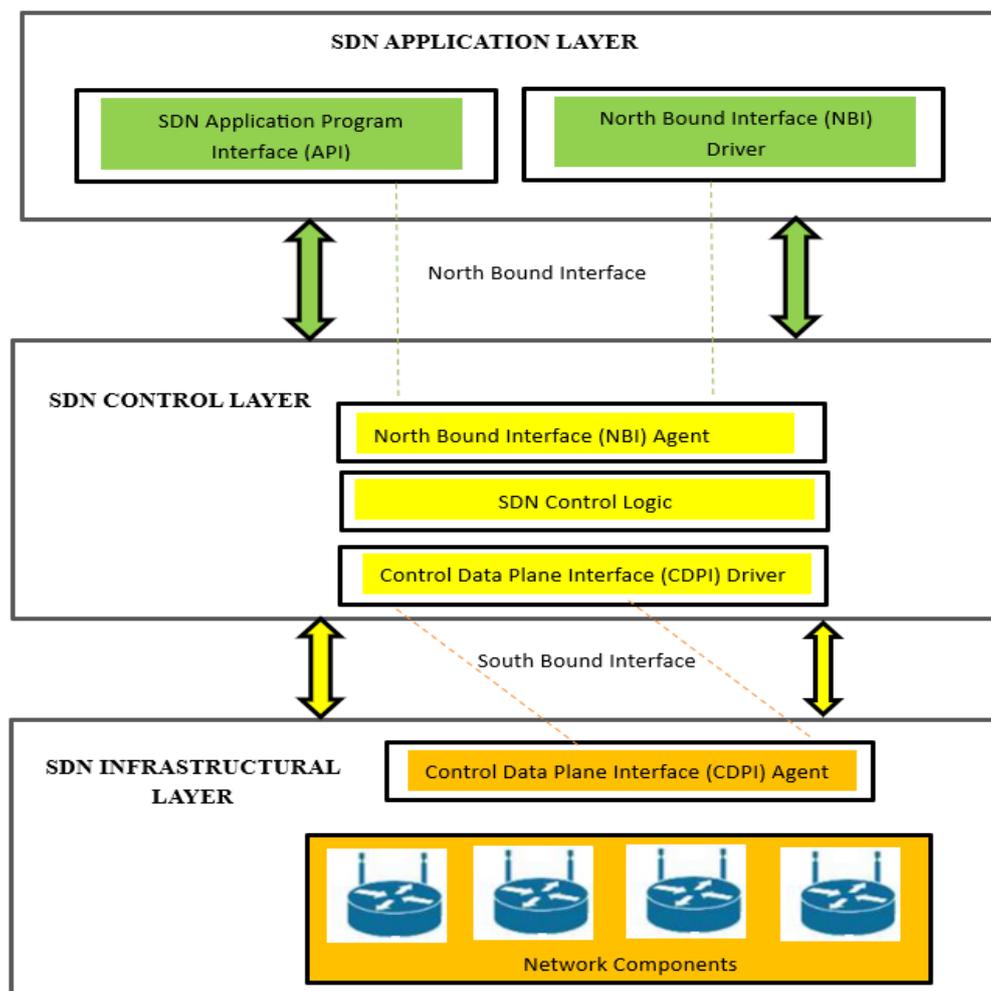


Figure 1. Layered structural model of SDN (software-defined networking).

Five important parameters exist to accomplish a complete view of SDN operation: the SDN application plane, control plane, data plane, northbound interface/northbound interface (NBI) agent, and southbound interface/control data plane interface (CDPI) [10,11]. The user interacts with the SDN network through the application layer by providing instructions to the controller of the SDN in simple logic, depending on which type of controller is selected to manage the flow of network components. The northbound interface is used to connect the application and control layers. With the help of the NBI agent, the controller in the second layer coordinates with the application layer and controls the network components (switches, routers, etc.) according to the instructions provided by the

designer in the application layer. The southbound-CDPI correlates with the control and data planes. The network component flow table is managed by an SDN controller using a CDPI agent. It informs the user of the flow result and QoS parameters with the help of NBI agents for further system data modulation.

Motivations and Significance of Research Technique

In this research, the proposed technique (DASLM) provides the following features:

- (a) By applying the DASLM approach, there is less end-to-end latency delay, maximum throughput, and less queuing delays by avoiding elephant flows, equal load management, and efficient use of bandwidth can be achieved.
- (b) Bandwidth enhancement is obtained using the proposed (DASLM) technique.
- (c) In the case of large networks (more switches, hosts, data center servers), the single controller could become loaded, exhausted, and lead to a single-point failure. The solution mentioned in different research techniques, as discussed in Section 2.1, was to use multiple controllers (in the master-slave version). The major issues in the implementation of these methods involve the following:
 - (1) The compatibility of two controllers.
 - (2) Providing the data center server access to both controllers.
 - (3) Switches flow managed by both controllers.

To overcome this problem, the large SDN network (i.e., servers in the network having to serve HTTP requests greater than thirty thousand per second and with more than two hundred HTTP requests in queue to be processed by the respective servers) is divided into the smaller SDN networks with the controller in a logically distributed controlled environment (to overcome the compatibility issue). Each controller is responsible for their (subdivided) SDN network.

- (d) The other major problem addressed in the proposed technique is the load balancing in the HTTP service provider servers by calculating the number of HTTP requests on each server and computing the server load. The HTTP request per second value (RPS) of each server is compared with the reference server load threshold (S_{LT}) value (as explained in Section 3.1), which is set to a level of 1000 (HTTP requests per second) in our case. If the number of HTTP requests on the particular server has reached the (S_{LT}) value, then that server is considered loaded and is removed from the available pool of servers in the SDN network, and no new HTTP request is assigned to that server until the RPS value decreases below the S_{LT} value range. The load is balanced by directing the flow of HTTP requests from the loaded server to other available servers on the following bases:

The new HTTP request flow is assigned to the server with a lower RPS load value than the list of available servers in the network.

(OR)

The new HTTP request flow is assigned to the server with less than 60 HTTP requests in the queue (considered a reference value in our case) in addition to the HTTP requests being processed by that server.

(OR)

The new HTTP request flow is assigned to the server with a quicker response time than all other available servers in the network. This task is accomplished by sending an ARP packet to the servers. The server that responds to the controller with less latency is then forwarded the new HTTP request flow.

However, the detail about the functioning of the proposed (DASLM) algorithm is mentioned in Section 3.1.

However, the paper structure summary is as follows:

Section 2 discusses the literature review and compares traditional load-balancing methods with the proposed (DASLM) algorithm. Section 3 explains the methodology of the proposed technique, lab setup details, and procedural steps. Section 4 discusses the

simulation results obtained in two portions: (1) without implementing the proposed algorithm (DASLM) on the SDN controller and (2) with implementing the proposed algorithm (DASLM) on the SDN controller. Each portion is simulated for two cases: (a) normal flow and (b) loaded flow, and QoS parameter comparison is performed with some traditional server load-balancing techniques to check the performance of the proposed (DASLM) algorithm. Section 5 summarizes the simulation result in conclusion with future directions.

2. Literature Review

2.1. Traditional Methods for Load Balancing in SDN Network

This section summarizes the theoretical results of the different research techniques for obtaining better QoS parameters. This portion is divided into three main categories to identify the research gap, as shown in Figure 2. The results are summarized as follows: (a) data flow (from the control plane to the data plane or vice-versa); (b) control flow (from the application layer to the control layer (and vice-versa) and the control plane to the data plane) [12]. The user interacts with the SDN network through the application layer by providing instructions to the controller of the SDN in simple logic, depending on which controllers are selected to manage the flow of network components.

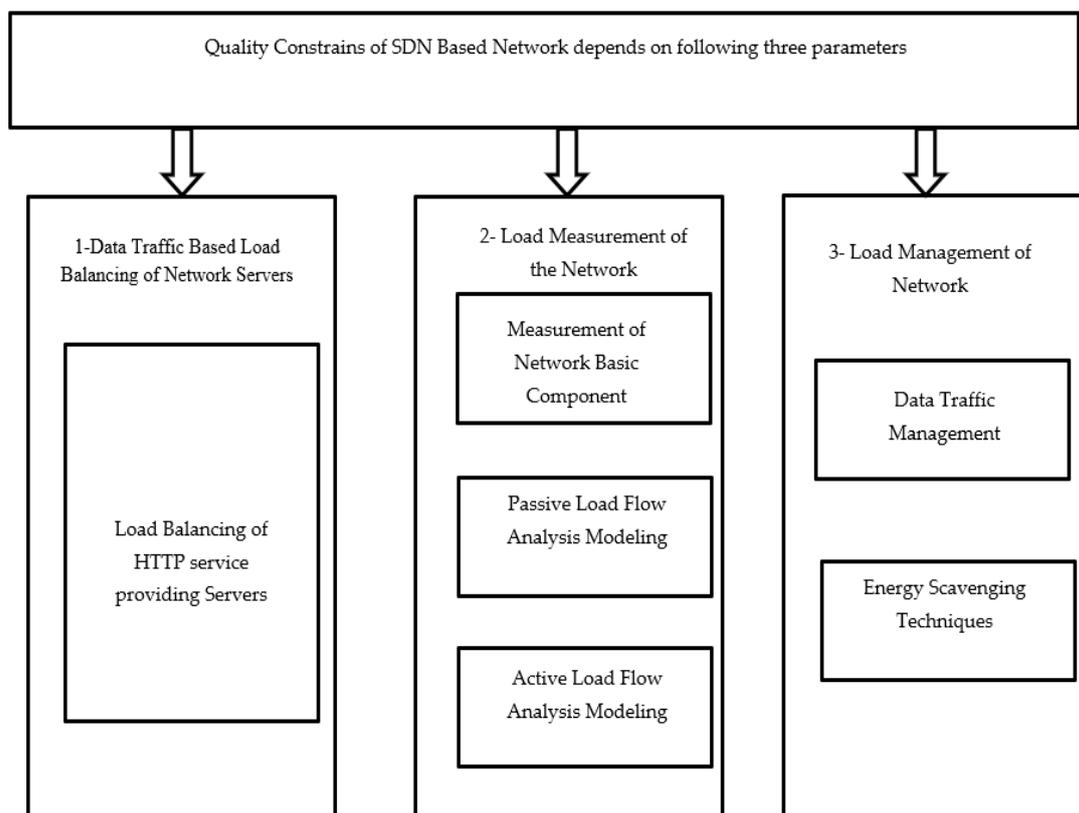


Figure 2. Structural view of literature review conducted in three main categories.

2.1.1. Load-Balancing Techniques of Network Servers

The network servers are connected in group form to fulfill the users' hypertext transfer protocol (HTTP) requests. However, servers are becoming exhausted due to the ever-increasing user request load. If server load management is neglected, the specific links will be overloaded, the HTTP request will be queued, an end-to-end delay will increase, and QoS will be deterred. This scenario will also lead to the complete network's exhaustion, eventually collapsing. Several QoS-oriented algorithms have been developed in modern-day research [1]. The authors of a research article [13] focused on the controller. Instead of using a single controller, they used master and slave versions. One controller looked after the flow, and the other managed the control signal; therefore, due to this technique, the

load was shared between two controllers. This resulted in reduced end-to-end delay and improved bandwidth utilization and throughput. However, this method added complexity regarding the compatibility of both controllers while transferring load and necessary control information. Another problem associated with this technique is the high energy consumption. In a previous study [14], load balancing was performed on network switches with an additional entry of data flow, so the data packets should have led to a specific server with less migration. The major problem was that if a particular server was requested more than other servers, that particular server would be loaded, so the technique mentioned above must be revised. In a previous study [15], the authors discussed a geo-based routing protocol in which the controller forwards the data packets to the servers closest to the switch so that there is a lower cost of migration. This leads to a problem: if the specific server is close to more switches in the network compared to the other servers, then that server will be loaded, and the QoS will be compromised. In a previous study [16], the static load-balancing algorithm was used, with data traffic categorized into two groups: (a) critical time traffic and (b) non-critical time traffic. First, the critical time traffic is routed if the server is loaded with critical time traffic flow, then the non-critical time traffic is discarded, which leads to a reduction in bandwidth utilization, and the overall throughput of the network is reduced. It is based on weighted Round-Robin method. In a research article [17], the authors added a programmable middle-box to assist with the SDN controller. It calculates the load on each server and guides the controller regarding server HTTP request handling and the packet drop ratio. This helps to share the load equally among the servers, but this leads to the additional complexity of programming the middle-box and controller. The middle-box and the controller should be compatible. In a previous study [18], the authors suggested using flux function calculations to find the switching weights routing cost from the server to the switch and then routing the data packets to the corresponding servers. The major problem is that if the routing cost of a specific server is low and is already loaded, then the above-mentioned network technique is inefficient. The researchers in [19–23] proposed that multiple controllers are the solution to QoS routing in SDN, but they introduce more complexity. The authors of [24,25] suggested that only the load can be balanced by adding weight to the data. Data-based load balancing was performed in a previous study [26]. A research article [27] discussed web server-based load balancing. In a research article [28], the authors compared different load-balancing techniques to determine the research gap for increasing QoS in telecom networks. In a research article [29], the authors combined a content delivery network (CDN) and SDN to enhance the quality of the network. In a research article [30], the authors suggested a traditional round ribbon technique for maximizing the availability of servers. In a research article [31], the authors implemented the dynamic and static load-balancing methods and drew their fruitful effects. In a research article [32,33], the authors implemented load balancing with SDN on Campus Networks (CN).

2.1.2. Measurement of Network Basic Component

SDN is a centralized, controlled software-based approach that provides an abstract view of the entire network arrangement and manages its flow using a controller. The controller obtains the topological information of the underlying network devices by running protocols, such as the Link Layer Discovery Protocol (LLDP) and Spanning Tree Protocol (STP) [34]. In these protocols, the controller broadcasts the message (pack-out) to all switches in the network. The switches that receive the message (pack-out) forward the message (forward-out) to all other switches directly connected to them. In this way, all switches receive broadcast messages. All switches respond to the controller with a message (pack-in) that includes information about the switch and the other switches to which this switch is directly connected. This is how the controller of SDN creates an abstract view of network topology and gives the whole network a picture for scalability purposes to the application layer using a northbound interface [35]. In a research article [6], the researchers mentioned that changing all of the network switches (legacy) to virtual switches is not

practically suitable, so it is better to make them operate in hybrid mode. The operation mode of the hybrid method is illustrated in Figure 3. In a research article [36], the method discussed (Prog-ME) provides a statistical flow graph of each node of the network, and the average load on a particular node can be seen using this method. In a research article [37], the technique discussed (Open-TM) was used to determine the average load on each node and provide better results than the above-mentioned method. A research article [38] addressed the technique (I-Stamp) used to calculate the flow of each node by creating a matrix. The results obtained using this method were far more accurate than those obtained using the aforementioned method. However, the problem with this method is that the number of flows (two or more) between two identical switches is given a single entry in the matrix; therefore, the overall load calculations with this method do not provide accurate results. In a research article [39], the traffic flow in data centers was performed using Top-of-Rack (ToR) switches which are challenging to implement in a network system. A research articles [40–42] discussed several flow methods to judge how many controllers must be required to fulfill the network administrator’s demand and avoid the controller side’s delay.

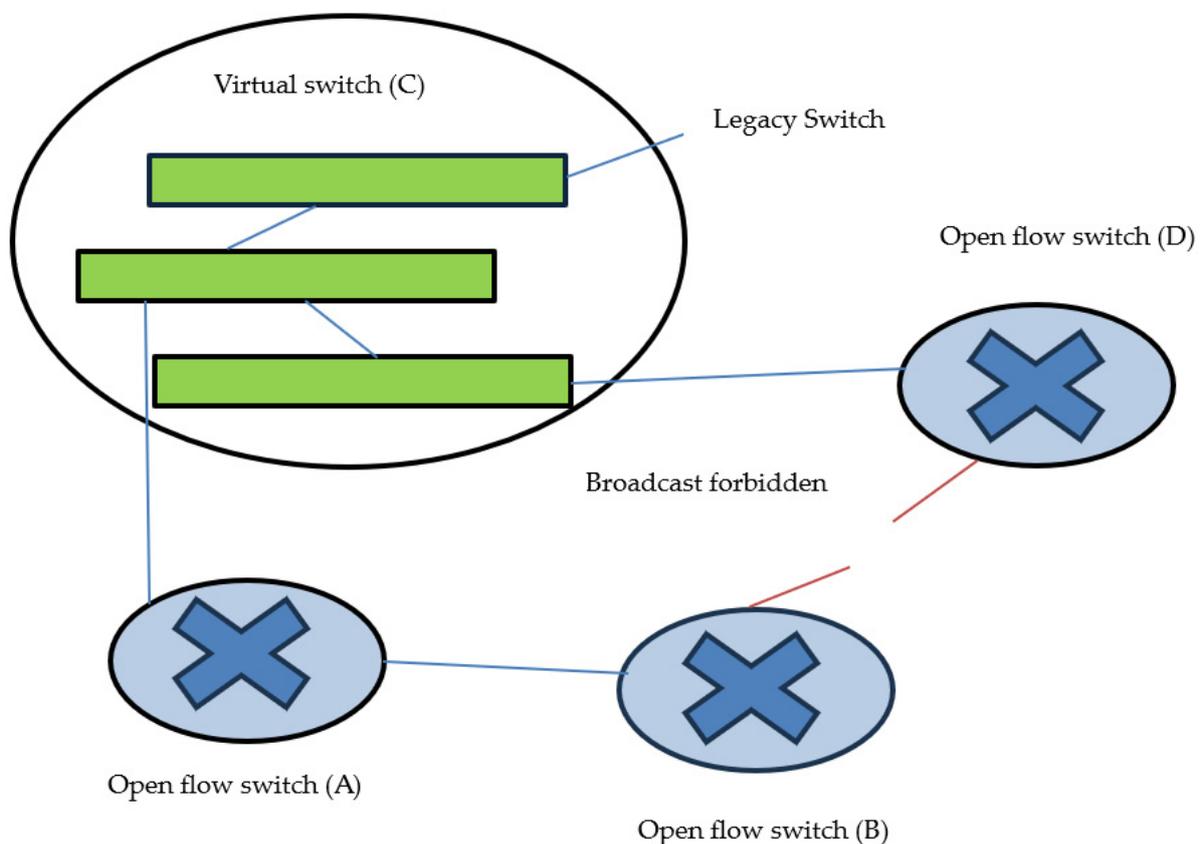


Figure 3. Open flow switches working with legacy switches in hybrid mode.

2.1.3. Passive Load Flow Analysis Modeling

A network load is divided into two main categories: (1) passive flow and (2) active flow. In this section, several techniques that are regarded as passive flow analyses are discussed. In a research article [43], the net flow developed by Cisco was discussed, providing an average graph of the network load instead of each flow. A research article [44,45] discusses S-flow and J-flow, indicating the con-troller about each network flow, which is excellent for a network with fewer network components and loads. Under loaded conditions, the S and J flow method can be irritating. In a research article [46], the polling method showed only the traffic graph if the user requests the northbound interface by instructing the application layer. A research article [47] discussed hash algorithm methods involving

extensive matrix calculations for traffic detection. In a research article [48], the dream physical project was introduced to determine the flows that must be mentioned by the controller to the application layer using NBI. Not every flow must be considered to avoid loading the controller.

2.1.4. Active Load Flow Analysis Modeling

Dynamic load flow analysis refers to traffic flow estimation by considering each flow rather than the average. The traffic estimation algorithm was developed in a previous study [49]; however, the researchers used fewer network components and a limited flow. This method can be applied to a specific scenario but not in real-time loaded conditions. A research article [50] selected the traffic detection model by considering only the active switch (switches with more data transfer are considered operational, and the rest are considered in sleep mode). This applies to small network systems. A research article [51] discussed several methods for determining flow calculation errors. In ref. [52], the veri-flow method is discussed, which can be placed between the control layer and data layer, providing the liberty that not all of the load flow must be managed by the controller but instead driven by the veri-flow. However, if there is no information in the veri-flow database regarding new data flow, the controller manages this new data traffic.

2.1.5. Data Traffic Management

The multipath routing scheme must be optimized to achieve better controllability of the data traffic. In a research article [53], one significant and straightforward problem related to Equivalent Multipath Routing (EMR) was the formation of elephant flows. An example formulation of the elephant flow is shown in Figure 4. In normal circumstances, for fast data transformation, the shortest path between two routers is selected for data transformation. However, if that path is loaded and no alternate route is chosen, the queue and delays will increase, and the overall throughput with bandwidth will decrease, resulting in reduced quality constraints. In refs. [54–57], several logics were developed to judge elephant flow scenarios. However, they involved extensive iteration of complex logic, which can produce several syntax errors and are challenging to manage. In ref. [58], the data were routed through different routers by considering the shortest path but with the addition of the timer. If the flow of data (packets) from one router to another occurs before the timer exists, the flow is considered normal. However, if the above-mentioned condition is not satisfied, the flow is regarded as an elephant flow, and the packet is rejected and requested to be sent again. In ref. [59], the priority was selected for each flow. The heavy priority flow is treated by timer management, and the rest is treated by a common EMR technique [60]. In refs. [61,62], link state optimization was adopted to overcome the aforementioned issues. If the shortest path is loaded, the data are shifted to the router with fewer packet forwarding requests under the supervision of the SDN controller.

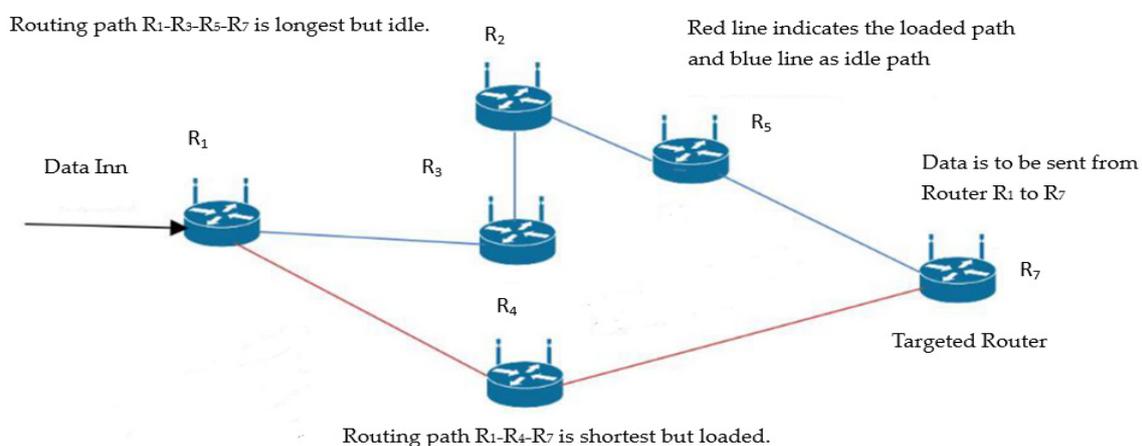


Figure 4. Scenario of elephant flow formation in multipath routing.

2.1.6. Energy Scavenging Techniques

Today, the necessity of the hour is to save energy because we are facing a severe energy crisis. In a research article [63,64], the overall energy consumption by the network components (routers, switches, etc.) is 5% of the total energy consumed in everyday life in developed countries and is assumed to be 10% by the end of the year 2020, owing to the revolutionized advancement in the field of networking. In a research article [65], several hit-and-trial-based methods are adopted to use fewer switches to save energy. In refs. [66,67], the energy scavenging technique makes the switches in standby mode; this factor saves 35% [68] of the energy.

2.2. Comparisons of the Proposed Algorithm (DASLM) with Traditional Load-Balancing Methods

Table 1 compares traditional load-balancing techniques with the proposed algorithm (DASLM).

Table 1. Comparison between proposed algorithm (DASLM) and traditional load-balancing techniques.

References	Improvement in Network with the Author's Technique	Limitation	Comparison with the Proposed Algorithm (DASLM)
S. Sathyanarayana et al. [69]	The authors combine the ant colony algorithm with the dynamic flow algorithm. The less-loaded server is found with the dynamic flow algorithm, and the shortest path to the less-loaded server is found using the ant algorithm technique.	In this paper, the latency is reduced. However, combining two algorithms and running them simultaneously extensively uses computer resources, memory, and bandwidth.	The proposed algorithm (DASLM) is a single active sensing dynamic algorithm that balances the load on HTTP servers in the SDN network by calculating their HTTP request load. If the HTTP request load of any server exceeds the range of the server load threshold (S_{LT}) value, the load is shifted to the server with less HTTP request load. However, if the above condition is not met, HTTP requests are forwarded to the server with a quicker response time. As a result, the transfer rate, available bandwidth, and throughput are increased, and there are no overhead issues.
H. Zhong et al. [70]	In this research paper, the HTTP request flow is managed based on server response time calculations.	Processing delays are reduced only.	The proposed algorithm (DASLM) not only balances the load on HTTP servers in the SDN network by calculating the response time by sending ARP packets but also selects the optimum server by (a) calculating RPS and comparing the RPS value with the reference threshold S_{LT} value and (b) finding the number of HTTP requests in the queue to be processed by the respective servers of the SDN network.
Hamed et al. [71]	The (HTTP request) load among different servers is balanced using the traditional Round-Robin method.	This method is simple and easy to implement and distributes the HTTP request load among different servers in sequential order. This method has greater limitations in large SDN networks with heavy data flow.	The proposed algorithm (DASLM) is more advanced than the method adopted for load balance [52]. In the proposed method, the optimum server for better managing HTTP request flow is selected based on response time calculation, calculation of RPS and comparison with threshold S_{LT} value, and finding the number of HTTP requests in the queue of respective servers to be processed.
Arahanashi et al. [72]	The (HTTP request) load among different servers is balanced by calculating each server's maximum available bandwidth.	The throughput and response time calculation is not considered.	The DASLM algorithm performs load balancing among different available servers based on (a) maximum available bandwidth (by calculating the server load), (b) response time, and (c) processing delays.
Kaur et al. [73]	The authors use a direct routing algorithm that directs the server's response to the host without passing through the load balancer. With this method, the author has claimed a decrease in latency.	The flow control is very much compromised.	In the DASLM algorithm, the RPS value of each server for every flow is calculated, and then, based on comparisons with S_{LT} value, the load is shared among different servers.
Kavana et al. [74]	The authors use a flood light controller, and the link path cost calculation is performed with the shortest path first.	HTTP request load is balanced among different available servers based on link cost optimization and no real-time traffic flow sensing.	DASLM performs real-time HTTP request flow sensing and distributes the HTTP request load among different available servers based on calculations performed for every flow.

Table 1. Cont.

References	Improvement in Network with the Author's Technique	Limitation	Comparison with the Proposed Algorithm (DASLM)
Hamed et al. [75]	Comparison of Raspberry-Pi-based network and the network formed on Mininet.	The result claimed by the authors is that the SDN-based network has better performance in server load balancing.	The DASLM is implemented in a Mininet environment with a POX controller.
S. Ejaz et al. [76]	The authors propose using two controllers (master and slave). All copies of files regarding flow management in the network are saved on the master controller so that if the controller fails, other controllers manage the flow.	A logically centralized environment requires tight synchronization.	DASLM proposes the use of a logically distributed environment.
H.Gasmelseed et al. [77]	In this study, the authors propose the use of two controllers. One controller controls the TCP flow, while the other manages the UDP flow and shares files at the end of every flow so that if the controller fails, other controllers manage the flow.	A logically centralized environment requires tight synchronization.	DASLM proposes the use of a logically distributed environment. In this arrangement, every controller manages the flow of their subdivided network.
N.T. Hai et al. [78]	The data traffic is distributed into two categories: (1) critical time traffic and (2) non-critical time traffic, and in the case of congestion, the critical time traffic is given priority.	Minimized data transmission with a greater packet drop ratio.	In the DASLM algorithm, every traffic flow is given equal importance, and real-time HTTP request load calculations manage the flow.
M.L.Chiang et al. [79]	In this research article, the authors use a flood light controller with dynamic load balancing to reach the under-utilized server among different servers available in the network. The HTTP request load is shifted to the server with less RPS (HTTP request per second) load.	No work is conducted regarding response time.	The DASLM algorithm has the advanced feature of computing the server response time and finding the number of HTTP requests in the queue to be processed by the respective server.
H.Zhong et al. [80]	This paper draws a comparison between static and dynamic scheduling algorithms.	The dynamic scheduling algorithm has better flow characteristics.	DASLM is the active sensing load-balancing algorithm that performs real-time calculations to distribute the HTTP request load uniformly among network servers.

3. Research Methodology

This section is further subdivided into two parts (the theoretical background and the methodology of the proposed technique).

3.1. Foundational Theoretical Background

Before proceeding to the implementation of the proposed research technique, the foundation theory of the proposed method in the flow steps is as follows:

The step-by-step procedure of the dynamic active sensing server load managing (DASLM) algorithm to obtain the above-mentioned goals is as follows, with a flow chart representation of each step:

Step 1

After establishing the SDN environment, the first step in the algorithm determines which controller is required depending on the selected network topology. In this research article, the single POX controller with the DASLM algorithm manages the load on user-defined network topology. However, using a single, centralized mode-based SDN controller

on large data center networks (DCNs) comparing several hundred thousand components or network types mentioned above will lead to more significant latency delays. Eventually, it could result in network failure due to data traffic overloading and bottlenecks [81] issues. The quality parameters in larger DCNs controlled by a single SDN controller will decrease significantly [82]. An experimental study is presented in a research article [83] in which significant DCN data traffic is controlled by a single NOX controller with low-quality parameters (i.e., latency of 10 ms). To combat the higher latency issue in larger SDN-based DCNs, the SDN controller should be used in either a logically centralized or logically distributed arrangement [84,85]. However, no universal technique or method provides the exact formulation to define when the Software-Defined Networking (SDN) is considered “large.” The following fundamental factors can be helpful to explain if the selected SDN-based network is small or large:

- (1) Network devices (routers, switches, etc.).
- (2) Network infrastructure.
- (3) QoS results extraction.
- (4) When the controller of an SDN-based network provides greater latency delays in HTTP request handling and indicates that the controller is not performing load management tasks properly. Example of a large SDN-based network:
 - (a) A network has a hundred thousand network devices (routers, switches, servers, etc.), a large number of concurrent HTTP requests generating end users, and multiple data centers.
 - (b) A network has hundreds of thousands of virtual machines, and their communication is managed through SDN-based applications.
 - (c) An SDN network provides services to many end-users covering a sizeable geographical area.

To overcome this problem illustrated in research articles [81–83], this research article proposes that the large SDN network be divided into the smaller SDN networks, with the controller in a logically distributed controlled environment (to overcome the compatibility issue). Each controller is responsible for their (subdivided) SDN network, and information regarding the tele-traffic details is shared through a communication link (wired or wireless). An explanation of step #1 in the DASLM algorithm is shown in Figure 5.

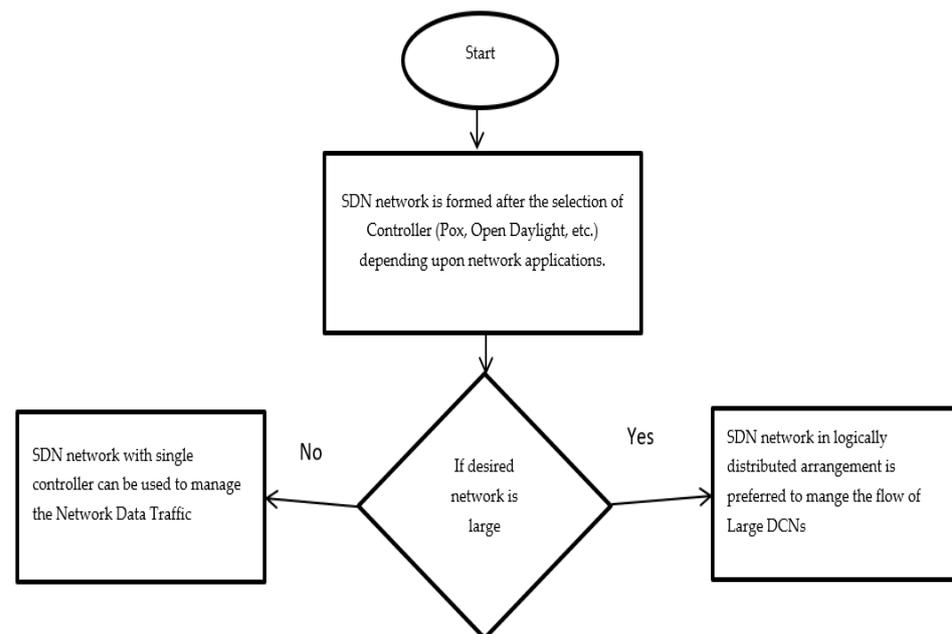


Figure 5. Flow diagram of step #1 in the DASLM algorithm.

Step 2

The primary purpose of the proposed technique is to prevent HTTP servers from overloading. After forming the SDN network, the next step in the algorithm is to calculate the number of HTTP requests on each server. Equations (1) and (2) are formulated in the DASLM algorithm to determine each server's HTTP requests/second (RPS).

$$\text{Number of requests} = J_{\text{step}} \times NPJ \times \text{Nuser/sec} \quad (1)$$

$$\text{HTTP requests/second (RPS)} = \frac{\text{Number of requests}}{\text{Total duration of time}} \quad (2)$$

RPS represents HTTP requests per second, J_{step} represents the number of steps in the journey, NPJ represents the number of HTTP requests per journey, and $N_{\text{users/sec}}$ represents the number of users per second. A journey is defined as an action performed by the user to originate an HTTP request to obtain the required information from a requested server. Sometimes, a simple HTTP request journey involves different steps corresponding to other web pages in addition to the requested web page. During this scenario, additional HTTP requests are generated. Under the influence of the DASLM algorithm, the SDN controller extracts each server's value of J_{step} , NPJ , $N_{\text{user/sec}}$, and RPS. Every server has a practical limit for handling HTTP request load; beyond that limit, the server performance decreases and can become unresponsive. In our proposed algorithm (DASLM), we have calculated the HTTP request per second load on each server. To tune the number of HTTP servers (four in our case) to handle the HTTP request load better, we have defined the maximum server load range as ("1000" HTTP requests per second) by using Equation (3).

$$\text{Server load range} = \frac{\text{total number of HTTP requests generated}}{\text{time in seconds}} \quad (3)$$

The HTTP traffic in an SDN-based network is calculated using the Wireshark tool in the Mininet. In our simulation model, "15,000" HTTP requests were generated during 15 s. Using Equation (3), we have set the server load range as ("1000" HTTP requests per second). Beyond this defined range, the server is considered loaded, and the HTTP request flow is assigned to the next available server in the network for load balancing. We have defined this server load range ("1000" HTTP requests per second) as the server load threshold (S_{LT}) value. If the number of HTTP requests on the particular server has reached this limit ("1000" HTTP requests per second), then that server is considered loaded. It is removed from the available pool of servers in the SDN network. No new HTTP request is assigned to that server until the RPS value decreases below the defined (S_{LT}) value.

No fixed theoretical limit applies universally to all SDN-based networks; one has to conduct load testing to find the most suitable server load range for efficient server load balancing in SDN-based networks. However, we have conducted load testing and monitored server performance (regarding network QoS parameters) to find the best server load theoretical value ("1000" HTTP requests per second in our case).

QoS parameter testing:

The "15,000" HTTP requests were generated from the randomly available twenty-three hosts and forwarded to the POX controller, whose task is to perform the HTTP request load balancing among four servers under the influence of the proposed algorithm (DASLM). As explained earlier, the maximum server load range defined as the server load threshold (S_{LT}) value is compared to the RPS value. In this case, to check the authenticity of the S_{LT} value, the network testing is performed in four parts. We have set the server load range to the value of (a) "2000" HTTP request per second, (b) "3000" HTTP requests per second, (c) "3750" HTTP requests per second, calculated from Equation (4), and (d) "1000" HTTP requests per second, calculated from Equation (3).

$$\text{Server load range} = \frac{\text{total number of HTTP requests generated}}{\text{total of HTTP server available in the network}} \quad (4)$$

Using the I-Perf utility, the network performance in QoS parameters is measured in all four cases for 15 s. The statistical data (QoS parameters) of all the above four cases are shown in Figure 6.

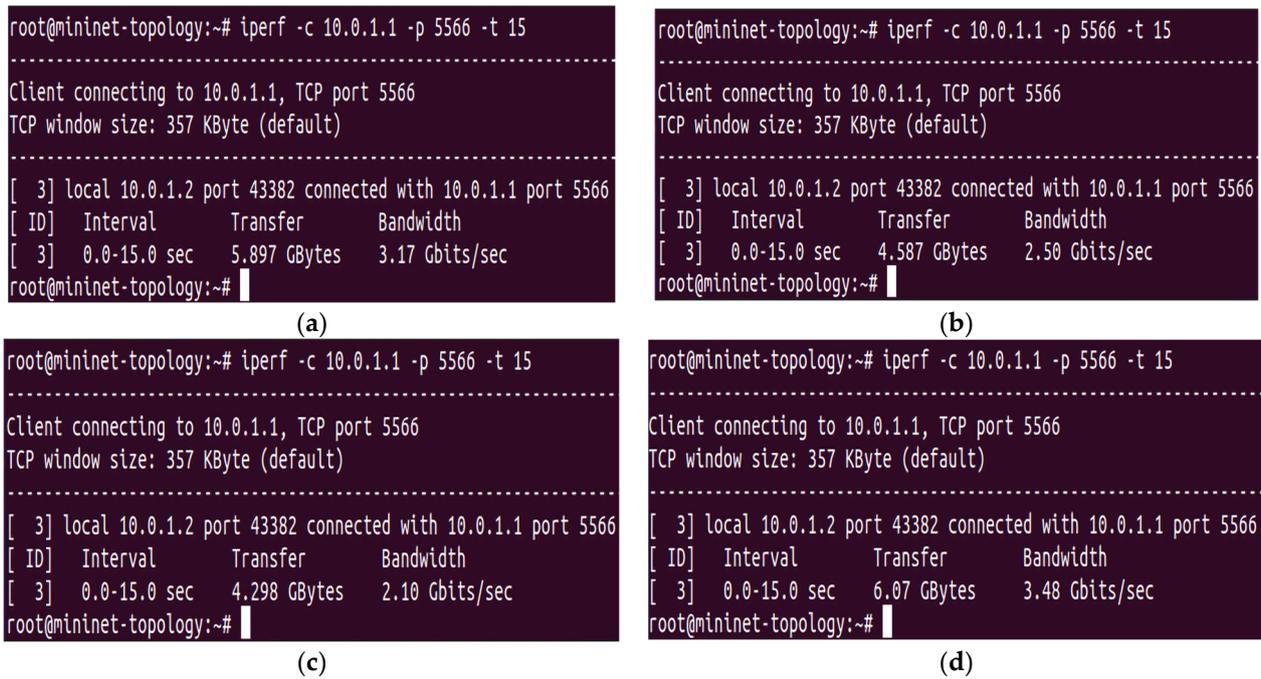


Figure 6. (a–d) QoS parameters of the interface between the controller and virtual for four load tests. (a) QoS parameters of the interface between the controller and virtual terminal using “2000” HTTP per second. (b) QoS parameters of the interface between the controller and virtual terminal using “3000” HTTP per second. (c) QoS parameters of the interface between the controller and virtual terminal using “3750” HTTP per second. (d) QoS parameters of the interface between the controller and virtual terminal using “1000” HTTP per second.

With reference to Figure 6, the QoS parameters are greater when the maximum server load value is selected using Equation (3). That is why the server load value defined as the S_{LT} value is chosen as “1000” HTTP requests per second. The QoS parameters (in terms of the transfer rate, throughput, and maximum available bandwidth) are summarized in Table 2. B_{am} represents the maximum available bandwidth, T_f is the transfer rate, and T_h is the throughput.

Table 2. Comparison of QoS parameters in all four tests obtained from I-Perf utility.

Load Testing	Time	B_{am}	T_h	T_f (in 15 s)
Test #4 (with 1000 HTTP requests per second)	0–15 s	3.48 Gbps	3.23 Gbps	6.07 Gbytes
Test #3 (with 3750 HTTP requests per second)	0–15 s	2.10 Gbps	2.29 Gbps	4.298 Gbytes
Test #2 (with 3000 HTTP requests per second)	0–15 s	2.50 Gbps	2.44 Gbps	4.587 Gbytes
Test #1 (with 2000 HTTP requests per second)	0–15 s	3.17 Gbps	3.14 Gbps	5.897 Gbytes

Conclusion (from four test cases):

Regarding QoS parameters obtained in Table 2, the proposed algorithm’s (DASLM) performance was better at defining the reference HTTP server load (S_{LT}) value (“1000” HTTP requests per second) and comparing it with the measured RPS value. If the number of HTTP requests on the particular server reaches this limit (“1000” HTTP requests per second), then that server is considered loaded. It is removed from the available pool of servers in the SDN network, and no new HTTP request is assigned to that server until the

RPS value decreases below the defined (S_{LT}) value. Further load balancing is performed as mentioned in step 3 of the proposed (DASLM) algorithm.

Step 3

Suppose the load traffic on a certain server exceeds the defined range of the S_{LT} value (as explained in step 2), in that case, the HTTP request is shifted to other servers for equal load sharing depending upon the fulfillment of the following sequential conditions:

IF (condition ==true)

{

The new HTTP request flow is assigned to the server with a smaller RPS load value than the list of available servers in the network.

ELSE

The new HTTP request flow is assigned to the server with less than 60 HTTP requests in the queue (considered a reference value in our case) in addition to the HTTP requests being processed by that server.

ELSE

The new HTTP request flow is assigned to the server with a quicker response time than all other available servers in the network. This task is accomplished by sending an ARP packet to the servers. The server responds to the controller with less latency, and the new HTTP request flow is forwarded to that server.

}

An explanation of step #2 and step #3 in the DASLM algorithm is shown in Figure 7.

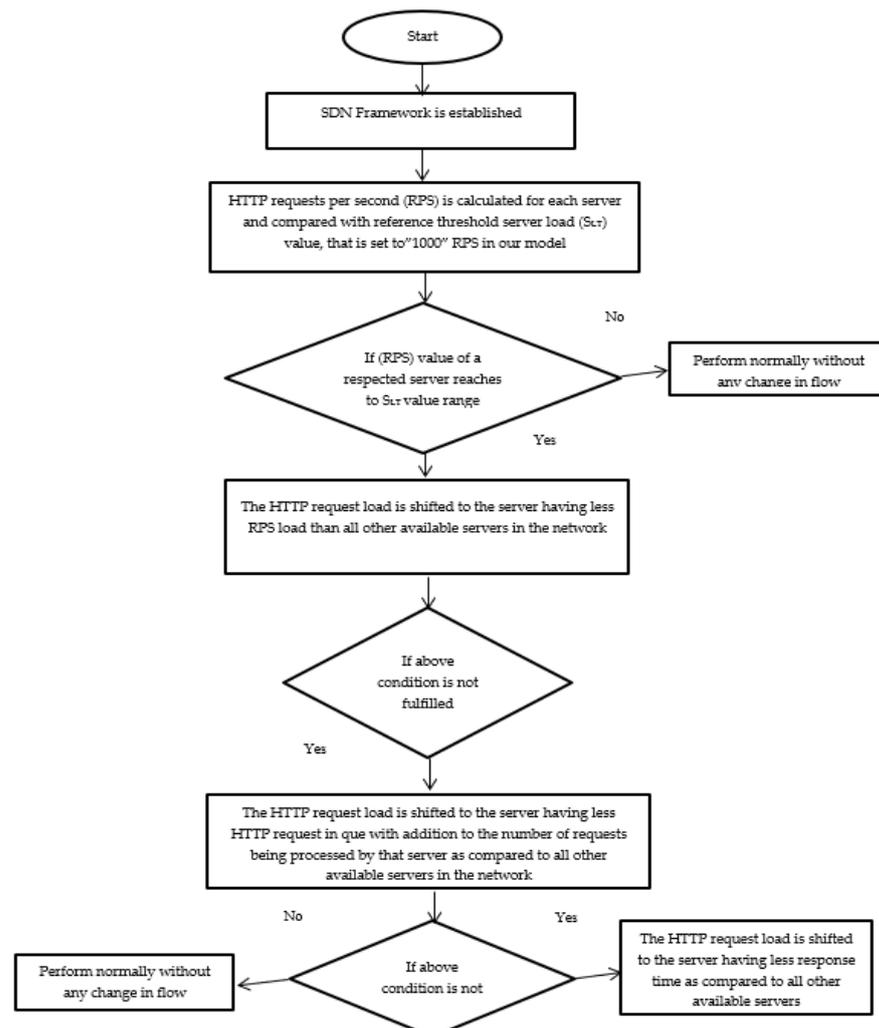


Figure 7. Flow diagram of steps 2 and 3 in the DASLM algorithm.

3.2. Procedural Steps

The simulation in this manuscript is performed in two portions: (1) without implementing the dynamic active sensing server load management (DASLM) algorithm on the SDN controller and (2) implementing the dynamic active sensing server load management (DASLM) algorithm on the SDN controller.

Portion 1:

- (a) SDN controller (POX) is first switched (up) to the running condition.
- (b) The network topology (shown in Figure 8) is drawn on the Mininet graphical interface or can be established by writing a command in the command line interface of Mininet.
- (c) Server load (in terms of HTTP requests) is calculated, and based on these calculations, the graph of the QoS parameters is obtained using the I-Perf and Gnu-plot utility.

Portion 2:

- (a) The controller (POX) is switched to active mode by running the DASLM algorithm (with details as mentioned in Section 3.1).
- (b) The network topology (shown in Figure 8) is drawn on the Mininet graphical interface or can be established by writing a command in the command line interface of Mininet.
- (c) Server load (in terms of HTTP requests) is calculated, and based on these calculations, the graph of the QoS parameters is obtained using the I-Perf and Gnu-plot utility.
- (d) The comparison is drawn between the QoS parameters results obtained in both portions (1 and 2). However, the QoS parameter results in portion 2 will be far superior to those obtained in portion 1 (the QoS result details are explained in Section 4).

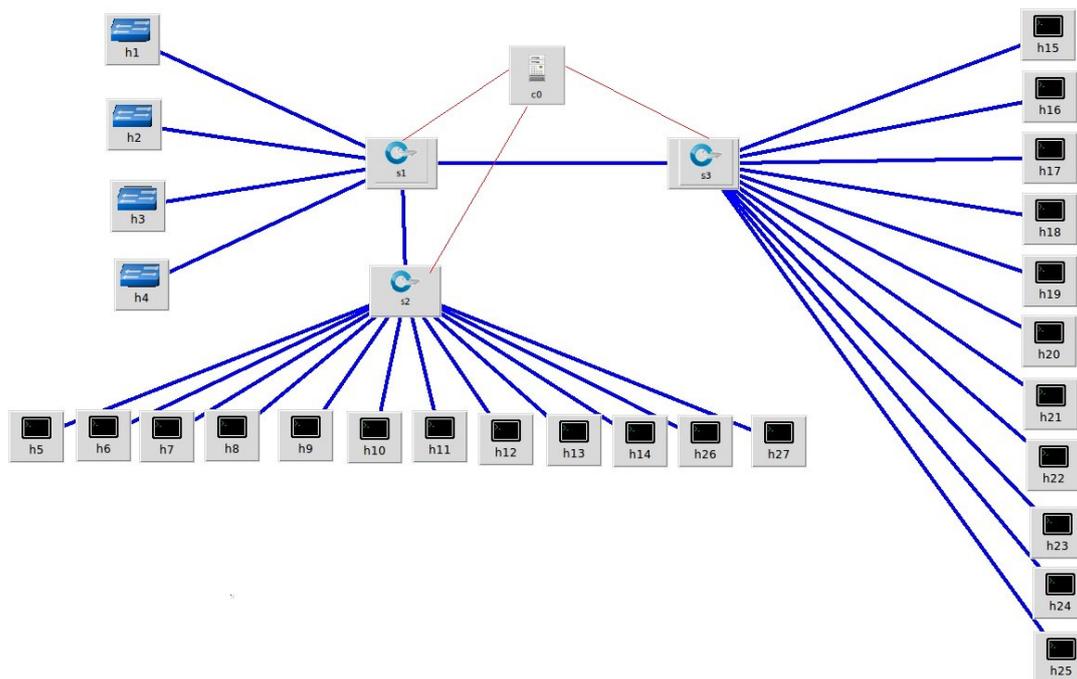


Figure 8. User-defined network topology on Mininet.

3.3. SDN-Based Environment (Lab Setup)

For SDN-based environment creation, we require the following:

- (a) Two cores i-7 (HP 15 Dw4029NE, Core i7, 12th Generation, 256 GB SSD, 1 TB HDD, 2 GB NVIDIA MX550 DOS), ten generations with 32 GB RAM each.
- (b) With three VMs (virtual machines) on each PC, one is used to run an SDN controller, one for the Mininet topology, and the other for the network performance graph.
- (c) Mininet is required to simulate the network along the I-Perf and J-Perf (required for QoS parameter measurement).

- (d) P-J-T graph and Gnu-plot utility convert text files in the simulated graph for QoS parameter analysis.
- (e) Wireshark tool (for network graphs).
- (f) POX Controller (scripted in Python—version: 3.11.4).

Table 3 represents the network parameters to be used in the simulation of a user-defined network in a Mininet environment.

Table 3. Network parameters to be used in the simulation of a user-defined network in a Mininet environment.

Parameters	Descriptions	Values
T in sec	Total simulation time in sec	0–15
B_{am}	Maximum available bandwidth in Gbps	Value to be calculated by I-Perf utility during both cases: (1) normal flow and (2) loaded flow
T_h	Throughput in Gbps	Value to be calculated by I-Perf utility during both cases: (1) normal flow and (2) loaded flow
T_f	Transfer rate in G-bytes	Value to be calculated by I-Perf utility during both cases: (1) normal flow and (2) loaded flow
S_{LT} value	Server load threshold value	1000 (RPS) is chosen as the reference value to compute the server load
L	Latency in ms	Value to be calculated by I-Perf utility during both cases: (1) normal flow and (2) loaded flow
RPS	Requests per second	150 RPS during case (1) normal flow and 15,000 during case (2) loaded flow
$\%T_f$	Percentage decrease in transfer rate	Value to be calculated by I-Perf utility during both cases: (1) normal flow and (2) loaded flow
$\%L$	Percentage increase in server load	Value to be calculated by I-Perf utility during both cases: (1) normal flow and (2) loaded flow
$\%B_{max}$	Maximum available bandwidth percentage	Value to be calculated by I-Perf utility during both cases: (1) normal flow and (2) loaded flow
$\%T_{af}$	Achievable transfer rate percentage	Value to be calculated by I-Perf utility during both cases: (1) normal flow and (2) loaded flow

4. Simulation Results and Discussion

The network selected for the simulation consisted of one POX controller with twenty-seven hosts and three switches. We converted the first four hosts (h_1 , h_2 , h_3 , and h_4) to HTTP servers (s_1 , s_2 , s_3 , and s_4) using the Python command. The network topology is illustrated in Figure 8.

We performed the simulation in two portions: (1) without implementing the dynamic active sensing server load management (DASLM) algorithm on the SDN controller and (2) implementing the dynamic active sensing server load management (DASLM) algorithm on the SDN controller with two cases, (A) normal flow and (B) loaded scenario. The results of the QoS parameters are obtained using the I-Perf utility. In the first portion of the simulation, in which no load management algorithm was loaded on the controller, the remaining twenty-three hosts were randomly used to generate 150 HTTP requests in normal flow, and 15,000 HTTP requests in the loaded scenario case, and they all were sent to four servers (s_1 , s_2 , s_3 , and s_4). In the second portion, the proposed algorithm technique (DASLM) was implemented on the POX controller using twenty-three hosts randomly; 150 HTTP requests in normal flow and 15,000 HTTP requests in the loaded scenario were generated and sent to the controller, which performs the load management task under the directions of the proposed technique.

4.1. (Case: Finding QoS Parameters of User-Defined Network Topology without the DASLM Algorithm)

4.1.1. Normal Flow

The maximum available bandwidth, throughput, and transfer rate were calculated for normal flow with 150 HTTP requests on each server. These calculations were performed using the I-Perf utility. The default IP addresses for servers (s_1 , s_2 , s_3 , and s_4) are (10.0.0.1), (10.0.0.2), (10.0.0.3), and (10.0.0.4), respectively. The statistical data (QoS parameters) fetched from these HTTP servers linked by the I-Perf utility for 15 s are shown in Figure 9.

```
-----
Client connecting to 10.0.0.1, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 19] local 10.0.0.5 port 47920 connected with 10.0.0.1 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-15.0 sec  33,2 GBytes 19,0 Gbits/sec
root@mininet-topology:~# iperf -c 10.0.0.1 -p 5566 -t 15
-----
Client connecting to 10.0.0.1, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 19] local 10.0.0.5 port 47922 connected with 10.0.0.1 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-15.0 sec  33,6 GBytes 19,3 Gbits/sec
root@mininet-topology:~#
```

QoS Parameters obtained from I-Perf Utility across server S1.

(a)

```
-----
Client connecting to 10.0.0.2, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 19] local 10.0.0.5 port 41922 connected with 10.0.0.2 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-15.0 sec  33,2 GBytes 19,0 Gbits/sec
root@mininet-topology:~# iperf -c 10.0.0.2 -p 5566 -t 15
-----
Client connecting to 10.0.0.2, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 19] local 10.0.0.5 port 41924 connected with 10.0.0.2 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-15.0 sec  34,4 GBytes 19,7 Gbits/sec
root@mininet-topology:~#
```

QoS Parameters obtained from I-Perf Utility across server S2.

(b)

```
[ 19] 0.0-15.0 sec  34,4 GBytes 19,7 Gbits/sec
root@mininet-topology:~# iperf -c 10.0.0.3 -p 5566 -t 15
-----
Client connecting to 10.0.0.3, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 19] local 10.0.0.5 port 47614 connected with 10.0.0.3 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-15.0 sec  32,5 GBytes 18,6 Gbits/sec
root@mininet-topology:~# iperf -c 10.0.0.3 -p 5566 -t 15
-----
Client connecting to 10.0.0.3, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 19] local 10.0.0.5 port 47616 connected with 10.0.0.3 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-15.0 sec  33,8 GBytes 19,4 Gbits/sec
root@mininet-topology:~#
```

QoS Parameters obtained from I-Perf Utility across server S3.

(c)

```
-----
Client connecting to 10.0.0.4, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 19] local 10.0.0.5 port 33488 connected with 10.0.0.4 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-15.0 sec  34,0 GBytes 19,4 Gbits/sec
root@mininet-topology:~# iperf -c 10.0.0.4 -p 5566 -t 15
-----
Client connecting to 10.0.0.4, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 19] local 10.0.0.5 port 33490 connected with 10.0.0.4 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-15.0 sec  34,5 GBytes 19,7 Gbits/sec
root@mininet-topology:~#
```

QoS Parameters obtained from I-Perf Utility across server S4.

(d)

Figure 9. (a–d) QoS parameters of HTTP servers (s_1 , s_2 , s_3 , and s_4) under normal flow without the DASLM algorithm through the I-Perf utility.

For a better understanding of the statistical data (in terms of transfer rate, throughput, and maximum available bandwidth) fetched across the four server links shown in Figure 9, they are summarized in Table 4, where B_{am} represents the maximum available bandwidth, T_h represents the throughput, and T_f represents the transfer rate.

Table 4. QoS parameters obtained from I-Perf utility (without DASLM).

List of HTTP Servers	Time in s	B_{am}	T_h	T_f (in 15 s)
Server#1	0–15 s	19.3 Gbps	17.92 Gbps	33.6 Gbytes
Server#2	0–15 s	19.7 Gbps	18.34 Gbps	34.4 Gbytes
Server#3	0–15 s	19.4 Gbps	18.0266 Gbps	33.8 Gbytes
Server#4	0–15 s	19.7 Gbps	18.4 Gbps	34.5 Gbytes

The Gnu-plot utility represents the maximum available bandwidth and transfer rate across the four HTTP servers in the line graphs. Figure 10 shows the QoS parameters (TCP data transfer rate T_f) across the servers (s_1 , s_2 , s_3 , and s_4).

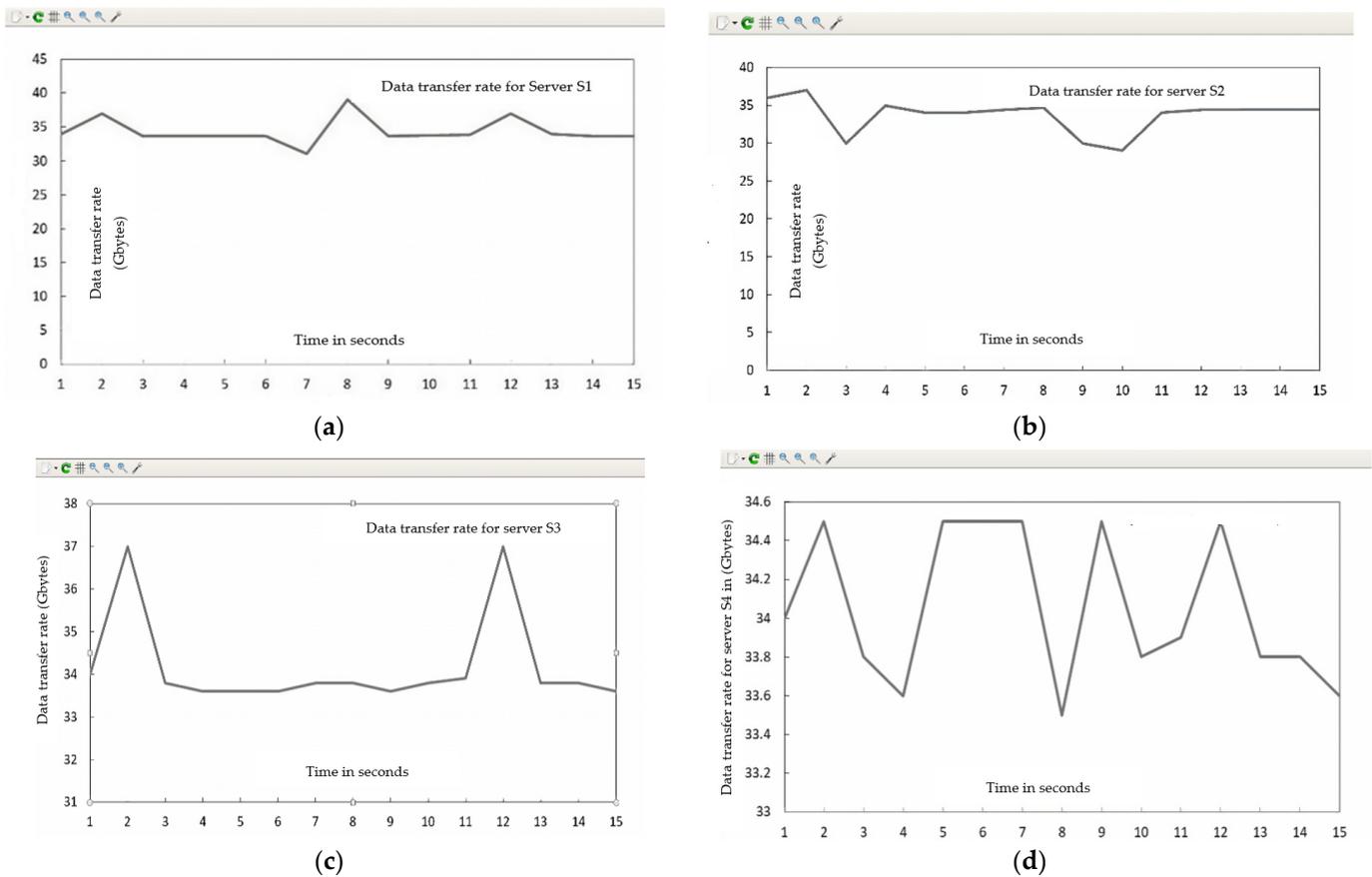


Figure 10. The QoS parameter (transfer rate = T_f) of HTTP servers under normal flow without implementation of DASLM algorithm. (a) (Transfer rate = T_f) of HTTP server (s_1). (b) (Transfer rate = T_f) of HTTP server (s_2). (c) (Transfer rate = T_f) of HTTP server (s_3). (d) (Transfer rate = T_f) of HTTP server (s_4).

Figure 11 shows the QoS parameters (maximum available bandwidth) across the servers (s_1 , s_2 , s_3 , and s_4).

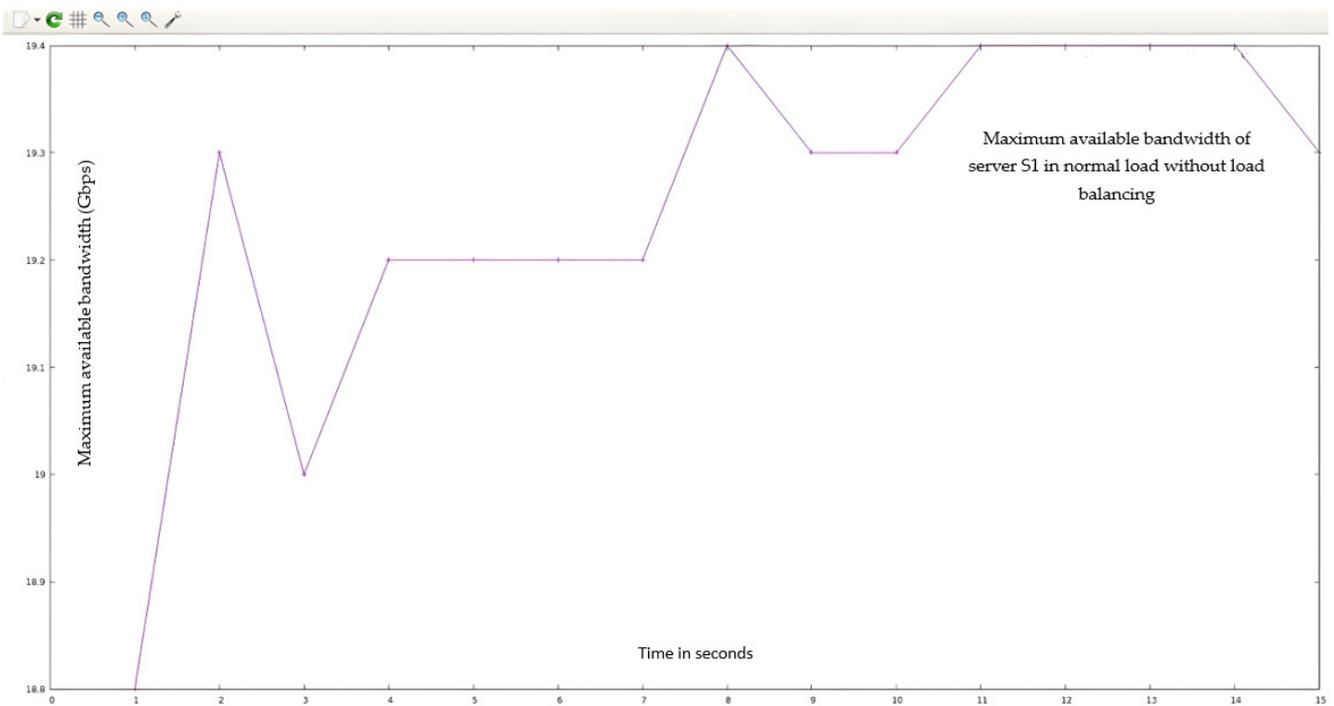
Summarizing the simulation results obtained from Case I (normal flow):

The maximum available bandwidth, throughput, and transfer rate were calculated for normal flow with 150 HTTP requests on each server. These calculations were performed by fetching data across four server (s_1 , s_2 , s_3 , and s_4) links for 15 s using the I-Perf utility and Gnu-plot. Referring to Figures 9 and 11, the Gnu-plot displays the maximum available bandwidth of the four servers in the form of a line graph. The maximum available bandwidths of servers s_1 , s_2 , s_3 , and s_4 with 150 HTTP requests are 19.3 Gbps, 19.7 Gbps, 19.4 Gbps, and 19.7 Gbps, respectively. Referring to Figures 9 and 10, the average transfer rate across the link of servers s_1 , s_2 , s_3 , and s_4 with 150 HTTP requests are 33.6 Gbytes, 34.4 Gbytes, 33.8 Gbytes, and 34.5 Gbytes, respectively. Equation (5) is used to determine the throughput across the four server links. The throughput of servers s_1 , s_2 , s_3 , and s_4 with 150 HTTP requests are 17.92 Gbps, 18.34 Gbps, 18.0266 Gbps, and 18.4 Gbps, respectively.

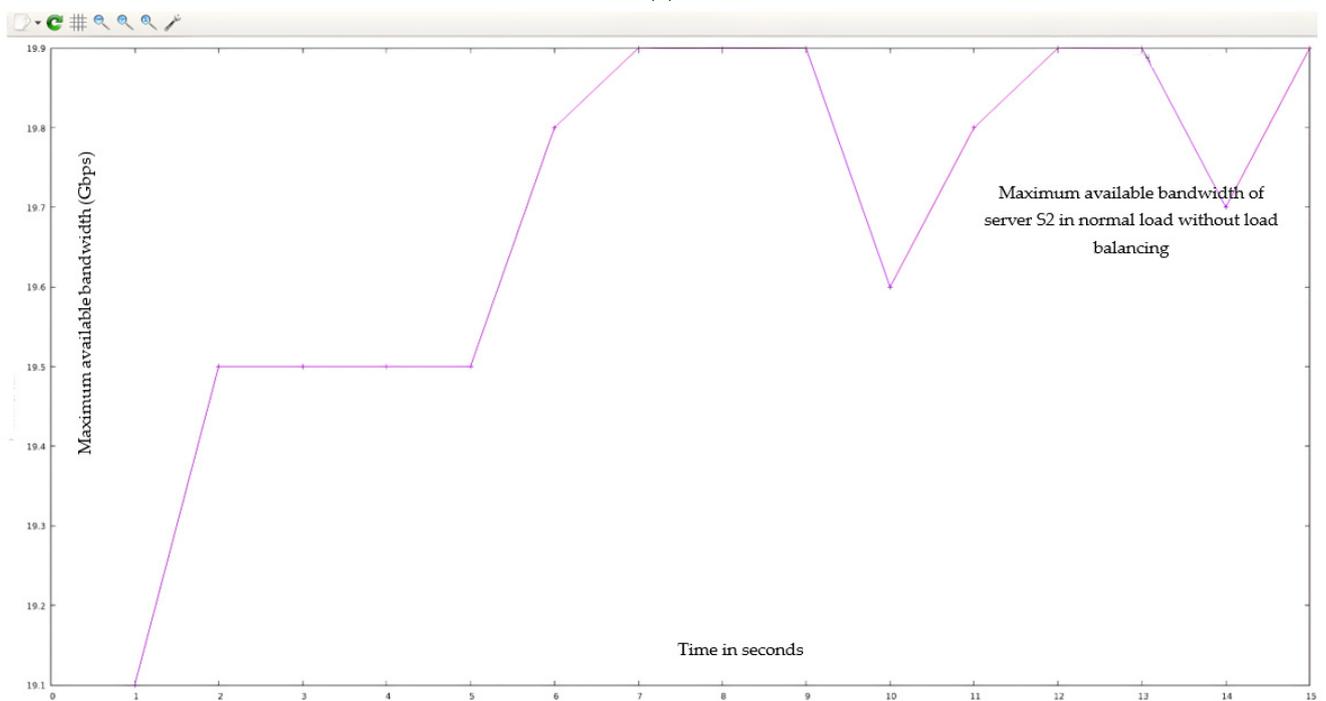
$$\text{Throughput (Gbps)} = \frac{\text{Data Transfer Rate in Gbytes}}{\text{Time in sec}} \quad (5)$$

The I-Perf utility makes one host a client and the other a server to which the HTTP request is forwarded. The I-Perf utility represents the statistical value of the network traffic at the server–client interface each time. The values that I-Perf represents are variable results (data) owing to the ratio of change in the network traffic across the server–client interface. In our case, as shown in Figures 10 and 11, a total of 150 HTTP requests were generated

and forwarded to each server (s_1 , s_2 , s_3 , and s_4) from the randomly available host in a total simulation time of 15 s. Therefore, the data represented by I-Perf in the real-time instance are variable because it takes 15 s for 150 HTTP requests to reach the servers. Hence, the data traffic ratio is different every instant (15 s), so the network traffic constantly changes.

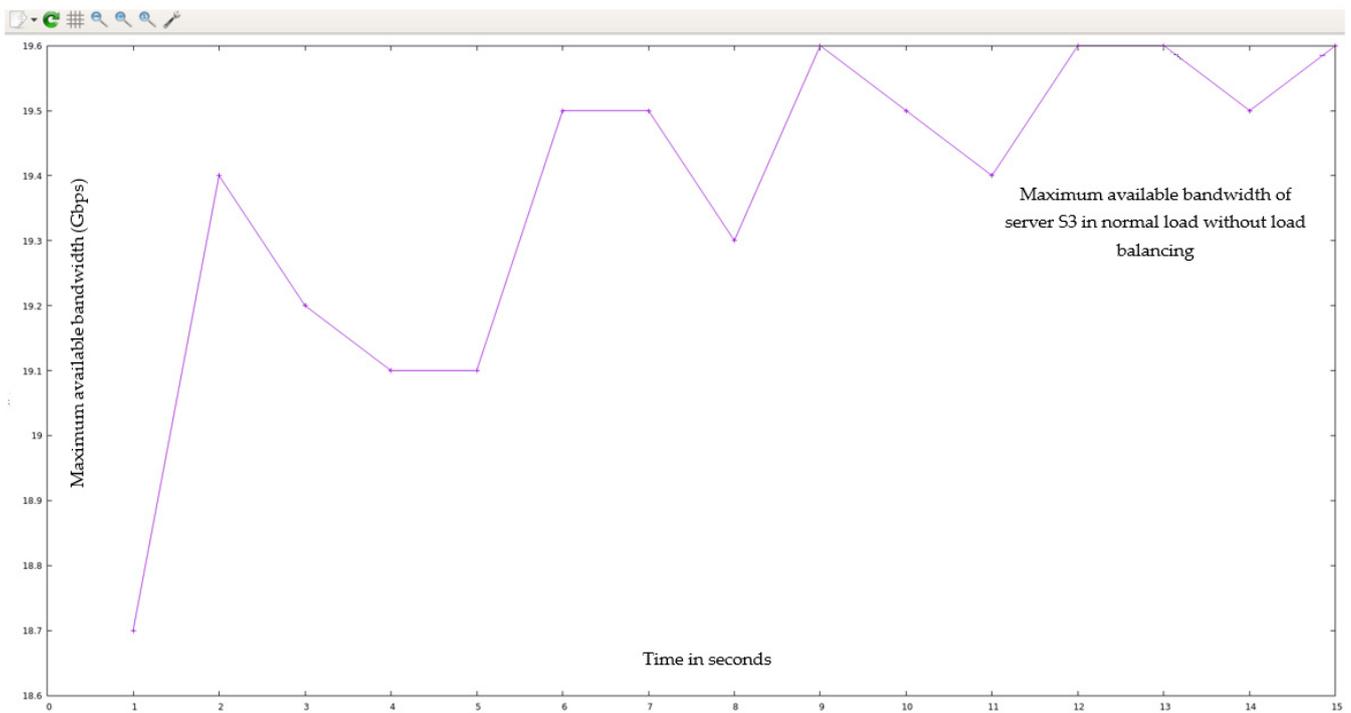


(a)



(b)

Figure 11. *Cont.*



(c)



(d)

Figure 11. (a) QoS parameters (max available bandwidth) of HTTP server (s_1) under normal flow without implementation of DASLM algorithm. (b) QoS parameters (max available bandwidth) of HTTP server (s_2) under normal flow without implementation of DASLM algorithm. (c) QoS parameters (max available bandwidth) of HTTP server (s_3) under normal flow without implementation of DASLM algorithm. (d) QoS parameters (max available bandwidth) of HTTP Server (s_4) under normal flow without implementation of DASLM algorithm.

4.1.2. Loaded Scenario

In this experiment, “15,000” HTTP requests were generated by randomly available twenty-three hosts (shown in Figure 8) and only directed to a specific server, s_2 ; no requests were generated for the other servers. In this case, the maximum available bandwidth, throughput, and transfer rate are calculated under loaded conditions for server s_2 (only). These calculations were performed using the I-Perf utility for 15 s. The QoS parameters (transfer rate, throughput, and maximum available bandwidth) fetched across the HTTP server s_2 under loaded conditions are summarized in Table 5, where B_{am} represents the maximum available bandwidth, T_h is the throughput, T_{avr} represents the time of arrival, L represents the latency, $\%T_f$ represents the percentage drop-in transfer rate compared to normal flow, $\%L$ represents the percentage increase in server load compared to normal flow, and T_f represents the transfer rate.

Table 5. Comparison of QoS parameters (with normal and loaded flow) obtained from I-Perf utility (without DASLM).

List of HTTP Servers	Time	B_{am}	T_h	T_f (in 15 s)	156 Packets T_{avr} (ms)	L (ms)	$\%T_f$	$\%L$
S2 (Normal Flow)	0–15 s	19.7 Gbps	18.34 Gbps	34.4 Gbytes	155,790.81	0.299	X	X
S2 (Loaded Scenario)	0–15 s	943 Mbps	0.88 Gbps	1.65 Gbytes	156,765	12	95.43%	95%

The statistical data (QoS parameters) fetched from these HTTP servers linked by the I-Perf utility for 15 s are shown in Figure 12.

```
Client connecting to 10.0.0.2, TCP port 5566
TCP window size: 85.3 KByte (default)
-----
[ 19] local 10.0.0.5 port 60526 connected with 10.0.0.2 port 5566
[ ID] Interval      Transfer      Bandwidth
[ 19] 0.0-15.0 sec  33.3 GBytes  19.1 Gbits/sec
root@mininet-topology:~# ^C
root@mininet-topology:~# iperf -c 10.0.0.2 -p 5566 -t 15
-----
Client connecting to 10.0.0.2, TCP port 5566
TCP window size: 85.3 KByte (default)
-----
[ 19] local 10.0.0.5 port 33144 connected with 10.0.0.2 port 5566
[ ID] Interval      Transfer      Bandwidth
[ 19] 0.0-15.0 sec  1.68 GBytes  964 Mbits/sec
root@mininet-topology:~# iperf -c 10.0.0.2 -p 5566 -t 15
-----
Client connecting to 10.0.0.2, TCP port 5566
TCP window size: 85.3 KByte (default)
-----
[ 19] local 10.0.0.5 port 35138 connected with 10.0.0.2 port 5566
[ ID] Interval      Transfer      Bandwidth
[ 19] 0.0-15.0 sec  1.65 GBytes  943 Mbits/sec
root@mininet-topology:~# █
```

Figure 12. QoS parameters of HTTP server (s_2) under a loaded scenario without DASLM through I-Perf utility.

As is evident from the data in Table 5, when all HTTP requests are sent to a specific server without an efficient load-balancing mechanism, the QoS parameters decrease, which results in the degradation of the network efficiency. The $\%L$ and $\%T_f$ explain that when all requests are directed to the specific server, the available bandwidth decreases (19.7 Gbps to 943 Mbps) owing to extra load (with only 4.78% available bandwidth and a load of approximately 95%) on the targeted server which is not shared by the other servers in

the network. This causes the targeted server to have a bottleneck condition, and the transfer rate decreases drastically to 1.65 Gbytes, with a decrease in transfer rate of 95.43% compared to normal flow. Implementing the proposed DASLM algorithm based on the aforementioned parameters can achieve high performance. The results are represented in Case II. Figure 12 shows the QoS parameters (maximum available bandwidth and transfer rate) across the targeted server s_2 . The B_{am} and T_f values obtained during the normal flow are considered references and compared to the loaded scenario, which is why the %L and % T_f values in the first row of Table 5 are marked (X). Equations (6) and (7) were used to find %L (percentage increase in server load compared to normal flow) and % T_f (percentage drop-in transfer rate compared to normal flow).

$$\%L = 100 - \left(\frac{100 \times (\text{avaible bandwidth under loaded case})}{\text{avaible bandwidth under normal flow}} \right) \quad (6)$$

$$\%T_f = 100 - \left(\frac{100 \times (\text{transfer rate under loaded case})}{\text{transfer rate under normal flow}} \right) \quad (7)$$

Summarizing simulation results obtained from Case I (loaded flow):

The maximum available bandwidth, throughput, and transfer rate were calculated for a loaded flow, with all “15,000” HTTP requests directed only to the targeted server (s_2) from randomly available hosts (as shown in Figure 8). These calculations were performed by fetching the data across the server (s_2) link for 15 s using the I-Perf utility and Gnu-plot. Referring to Figures 12 and 13, the Gnu-plot displays the maximum available bandwidth and transfer rate across the server (s_2) in a line graph. The maximum available bandwidth of the server (s_2) with “15,000” HTTP requests is decreased from 19.7 Gbps to 943 Mbps. The transfer rate across the link of server S_2 is reduced from 34.4 Gbytes to 1.65 Gbytes. Equation (5) was used to find the throughput across the s_2 links. The throughput of the server (s_2) is also decreased from 18.34 Gbps to 0.88 Gbps. As no load-balancing algorithm technique is applied to the controller of the SDN, the server load on s_2 is increased up to 95% (calculated from Equation (6)). The drop-in transfer rate of the server (s_2) is about 95.43% (calculated from Equation (7)). The ping command was used to find processing delays and latency. The arrival times of 156 packets were calculated. The processing delay is 974.19 ms in the loaded scenario. The latency increases to 12 ms, as mentioned in Table 5.

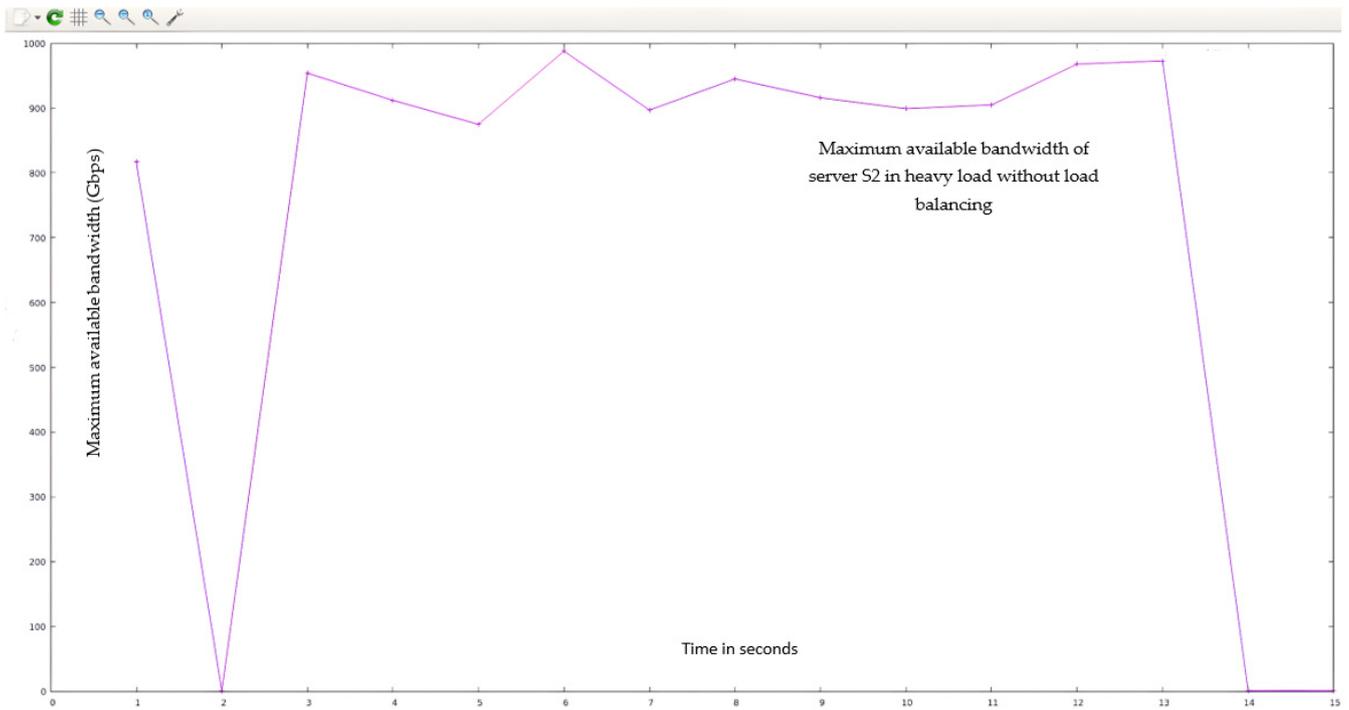
4.2. Case II: Finding QoS Parameters of User-Defined Network Topology with the Implementation of the DASLM Algorithm on an SDN Controller

This portion was divided into two parts to better interpret the results: (1) normal flow and (2) loaded scenario. When the DASLM algorithm script is loaded onto the SDN controller, it acts as a loadmaster and distributes the HTTP request from the host to a pool of available network servers. There is a slight difference in case II compared to the previously discussed case I. Here, HTTP requests are directed to the SDN controller, which performs a load-managing task under the instruction of the DASLM.

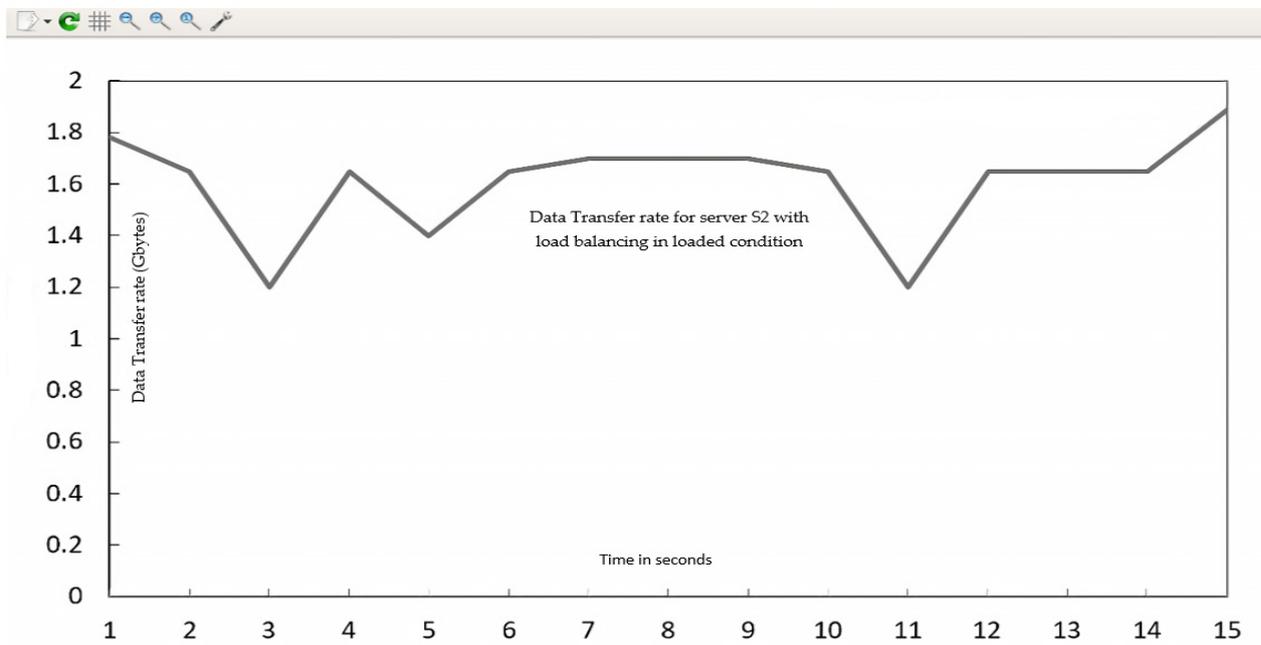
4.2.1. Normal Flow

In the case of a normal flow, “150” HTTP requests are directed to the controller, which, under DASLM, fulfills the user HTTP requests. The maximum available bandwidth, throughput, and transfer rate were calculated using the I-Perf utility for 15 s. Here, virtual traffic is generated from one virtual machine, and the controller is designed on the other. As per previous practice, “150” HTTP requests were sent from the virtual machine to the controller loaded with the DASLM algorithm script. The controller IP address is 10.0.1.1, whereas the virtual machine on which the virtual traffic is generated is 10.0.1.2. For a better understanding of the data (in terms of transfer rate, throughput, and maximum available bandwidth) fetched across the link connecting the controller under DASLM and the virtual machine, where virtual traffic is generated as HTTP requests, they are

summarized in Table 6, where B_{am} represents the maximum available bandwidth, T_h represents the throughput, and T_f represents the transfer rate.



(a)



(b)

Figure 13. (a) QoS parameters (maximum available bandwidth) of targeted server s_2 under a loaded scenario without DASLM. (b) QoS parameter (transfer rate T_f) of HTTP server (s_2) under a loaded scenario without implementation of DASLM.

Table 6. QoS parameters obtained from I-Perf utility with DASLM.

Interface	Time in s	B_{am}	T_h	T_f (in 15 s)
The link between the controller and the HTTP request generator virtual machine	0–15 s	4.02 Gbps	3.744 Gbps	7.02 Gbytes

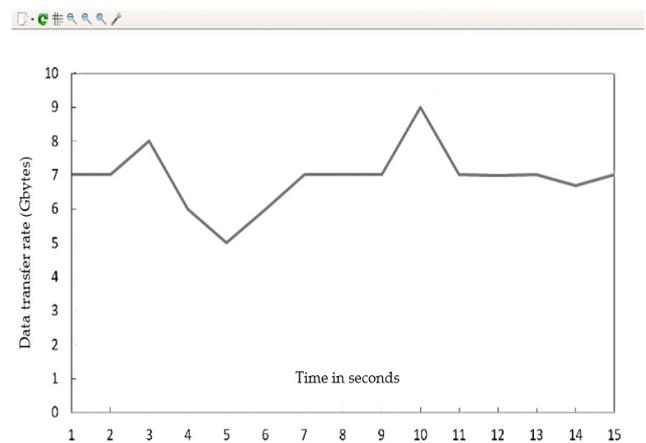
The Gnu-plot represents the maximum available bandwidth and transfer rate in the line graphs. Figure 14 illustrates the QoS parameters (data transfer rate and maximum available bandwidth).

```

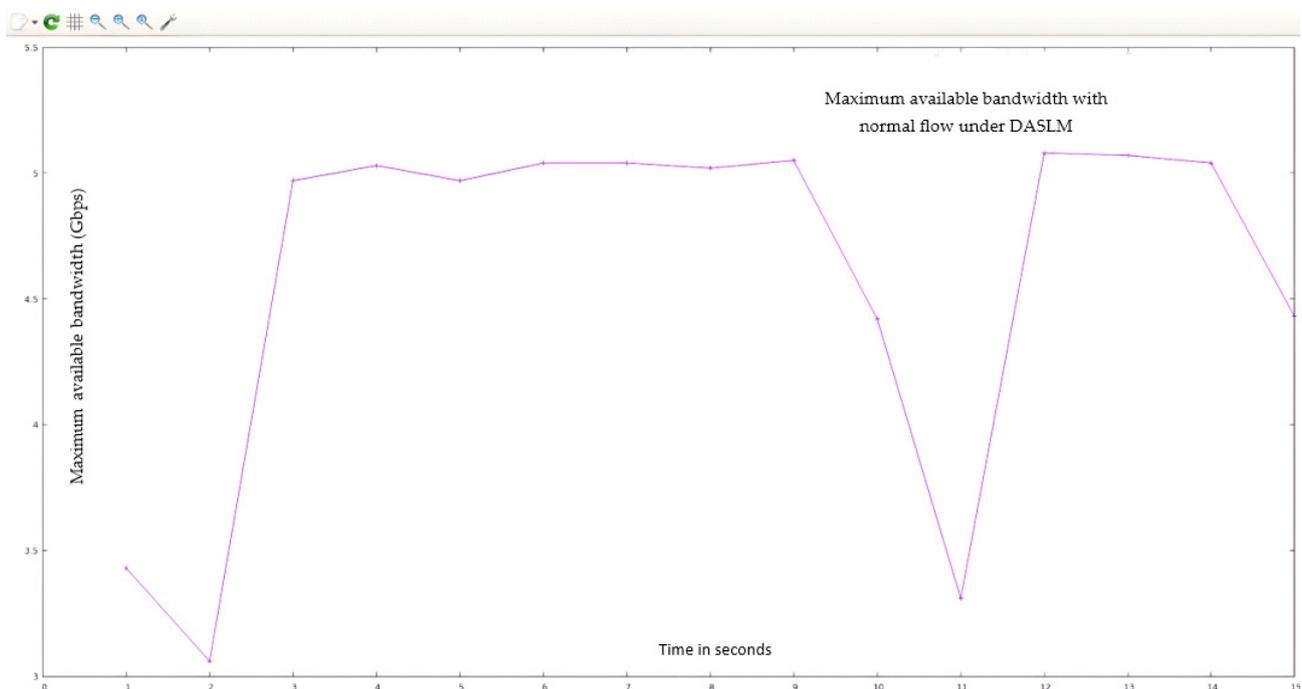
root@mininet-topology: ~
File Edit View Search Terminal Help
root@mininet-topology:~# iperf -c 10.0.1.1 -p 5566 -t 15
-----
Client connecting to 10.0.1.1, TCP port 5566
TCP window size: 357 KByte (default)
-----
[ 3] local 10.0.1.2 port 43382 connected with 10.0.1.1 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-15.0 sec  7.02 GBytes 4.02 Gbits/sec
root@mininet-topology:~#

```

(a)



(b)



(c)

Figure 14. (a–c) show the QoS parameters under normal flow with the implementation of the DASLM algorithm. (a) represents statistical information about QoS parameters on the Mininet terminal. (b) represents the transfer rate (T_f). (c) represents the maximum available bandwidth.

Summarizing simulation results obtained from Case II (normal flow):

In the case of a normal flow, 150 HTTP requests are directed to the controller, which, under DASLM, fulfills the user HTTP requests. The maximum available bandwidth and transfer rate calculations were performed using the I-Perf utility for 15 s. Referring to Figure 14, the Gnu-plot displays the maximum available bandwidth and transfer rate across the link between the controller and virtual machine in a line graph. The maximum available bandwidth is 4.02 Gbps. The transfer rate is 7.02 Gbytes. Equation (5) was used to find the throughput across the link between the DASLM-based controller and the virtual machine. The throughput is 3.744 Gbps.

4.2.2. Loaded Scenario

In this case, “15,000” HTTP requests are generated on the interface between the controller and the virtual machine. The maximum available bandwidth, throughput, and transfer rate were calculated under loaded conditions. These calculations were performed using the I-Perf utility for 15 s. Summary of statistical data obtain from Iperf utility is shown in Figure 15.

```

root@minine
File Edit View Search Terminal Help
-----
Client connecting to 10.0.1.1, TCP port 5566
TCP window size: 85.3 KByte (default)
-----
[ 3] local 10.0.1.2 port 34384 connected with 10.0.1
[ ID] Interval      Transfer      Bandwidth
[ 3] 0.0-15.0 sec  6.07 GBytes  3.48 Gbits/sec
root@mininet-topology:~# iperf -c 10.0.1.1 -p 5566 -t

```

Figure 15. QoS parameters of the interface between the controller and virtual machine under DASLM with the loaded scenario.

The QoS parameters (in terms of the transfer rate, throughput, and maximum available bandwidth) under loaded conditions are summarized in Table 7, where B_{am} represents the maximum available bandwidth, T_h is the throughput, T_{avr} is the time of arrival, L is the latency, $\%T_f$ represents the percentage drop-in transfer rate as compared to normal flow, $\%L$ represents the percentage increase in server load compared to normal flow, and T_f represents the transfer rate. Equations (8) and (9) are used to find the maximum available bandwidth percentage ($\%B_{max}$) and achievable transfer percentage ($\%T_{af}$), respectively.

$$\%B_{max} = \left(\frac{100 * (\text{available bandwidth under loaded case})}{\text{available bandwidth under normal flow}} \right) \quad (8)$$

$$\%T_{af} = \left(\frac{100 * (\text{transfer rate under loaded case})}{\text{transfer rate under normal flow}} \right) \quad (9)$$

As is evident from the data in Table 7, by implementing the proposed DASLM algorithm based on the parameters mentioned above, very high-performance QoS parameters are achieved when all HTTP requests (“15,000” in our case) are directed to the controller under DASLM, the available bandwidth increases from 943 Mbps to 3.48 Gbps, along with an increase in the transfer rate from 1.65 Gbytes to 6.07 Gbytes. $\%T_f$ represents the percentage drop-in transfer rate compared with the normal flow, and $\%L$ represents the percentage increase in server load. The factors $\%L$ and $\%T_f$ are also reduced from 95% to 13.43% and 95.43% to 13.53%, respectively, compared to case I without DASLM implementation. The latency and packet arrival time are also reduced, resulting in a decreased processing delay. With the implementation of the DASLM algorithm, efficiency in terms of maximum available bandwidth is increased from 4.786% (in case I) to 86.57%. The transfer efficiency with throughput also rose from 4.65% to 86.47%.

Table 7. Comparison of QoS parameters obtained from case I and case II using I-Perf utility.

Interface/Servers	Time	B_{am}	T_h	T_f (in 15 s)	Available Bandwidth Percentage	Achievable Transfer Rate (%)	156 Packets T_{avr} (ms)	L (ms)	$\%T_f$	$\%L$
(Normal Flow) with DASLM	0–15 s	4.02 Gbps	3.744 Gbps	7.02 Gbytes	X	X	790.81	0.2	X	X
(Loaded Scenario) with DASLM	0–15 s	3.48 Gbps	3.23 Gbps	6.07 Gbytes	86.57%	86.47%	865.67	0.87	13.53%	13.43%
(Loaded Scenario) without DASLM	0–15 s	943 Mbps	0.88 Gbps	1.65 Gbytes	4.78%	4.65%	156,765	12	95.43%	95%

The B_{am} and T_f obtained during the normal flow are considered references, and their values were compared to the loaded scenario, which is why $\%L$ and $\%T_f$ values in the first row of Table 7 are marked (X). The Gnu-plot utility determines the maximum available bandwidth, throughput, and transfer rate graphs. Figure 16 represents the QoS parameter (maximum available bandwidth). Figure 17 illustrates the QoS parameter (transfer rate).

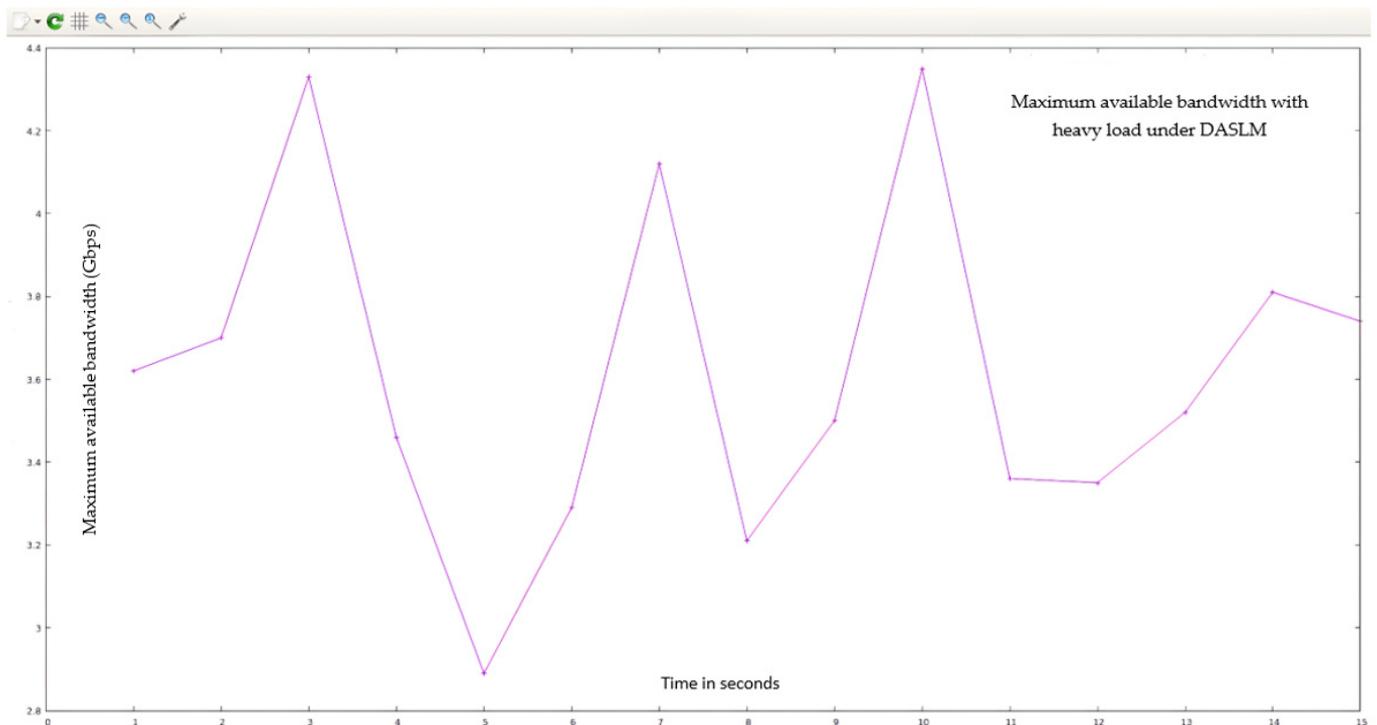


Figure 16. Represents the QoS parameter (maximum available bandwidth) with the loaded scenario with DASLM.

Summarizing simulation results obtained from Case II (loaded flow):

In the case of a loaded flow, “15,000” HTTP requests are directed to the controller, which, under DASLM, fulfills the user HTTP requests. The maximum available bandwidth, throughput, and transfer rate calculations were performed using the I-Perf utility for 15 s. Referring to Figures 16 and 17, the Gnu-plot displays the maximum available bandwidth and transfer rate across a link between the controller and the virtual machine in the form of a line graph. The maximum available bandwidth is increased to 3.48 Gbps, compared to the results obtained in Section 4.1.2. The transfer rate is also increased up to the level of 6.07 Gbytes in comparison with the simulation results of Section 4.1.2. Equation (5) was used to find the throughput across the link between the DASLM-based controller and the virtual machine. The throughput is also increased up to the level of 3.23 Gbps. Table 7 shows that with the implementation of the proposed technique, $\%L$ (calculated from Equation (6)) has decreased from 95% (without DASLM implementation) to 13.43% (with DASLM implementation), and $\%T_f$ (calculated from Equation (7)) has also decreased from

95.43% to 13.53%. Ping command was used to determine the latency. The latency decreased from 12 ms (without DASLM implementation) to 0.87 ms (with DASLM implementation).

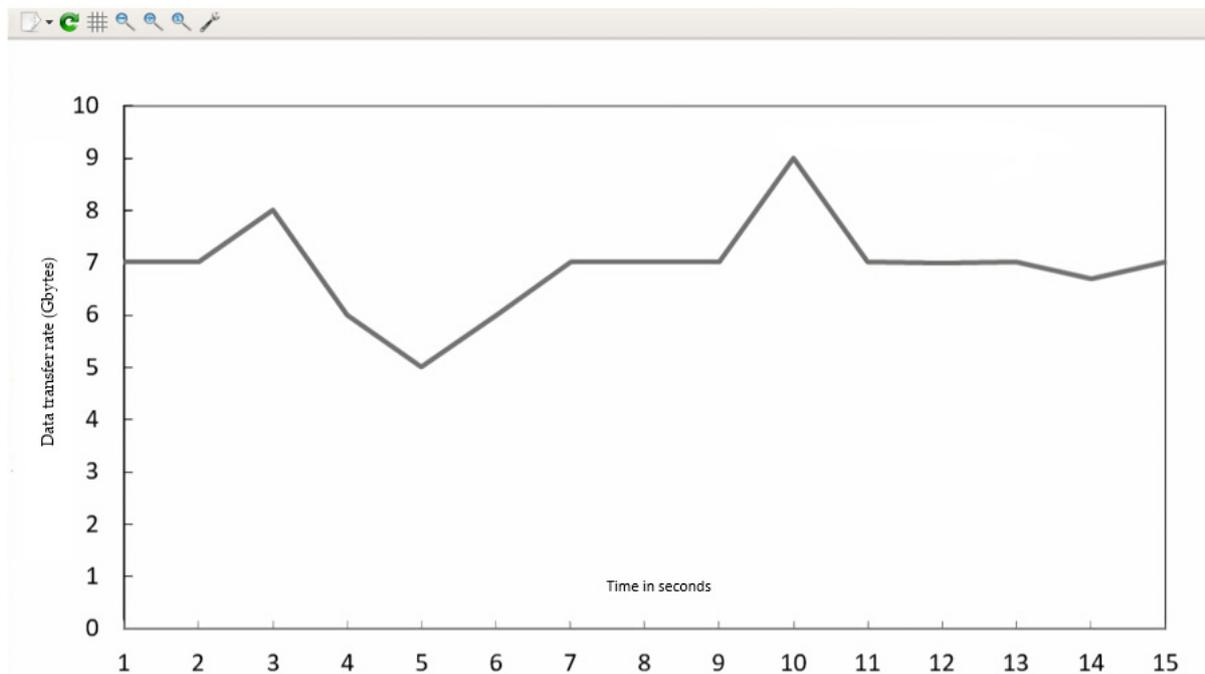


Figure 17. Represents the QoS parameter (transfer rate) with the loaded scenario with DASLM implementation.

4.3. Comparative Analysis of DASLM with Traditional Server Load-Balancing Methods

To prove the effectiveness of the proposed (DASLM) algorithm, the comparative analysis is conducted with the method discussed in research articles [70,71]. Table 1 explains the fruitful effects of the proposed algorithm (DASLM) compared to the other research methods. The two test cases (Case A and Case B) were conducted to judge the performance of the proposed algorithm. In Case A, the research method of article [70] (using the least server response method) is applied to the SDN controller for server load balancing, and QoS parameters are extracted. In Case B, the research method of article [71] (using the traditional Round-Robin method) is used for the SDN controller for server load balancing, and QoS parameters are extracted.

Case A (using the least server response method):

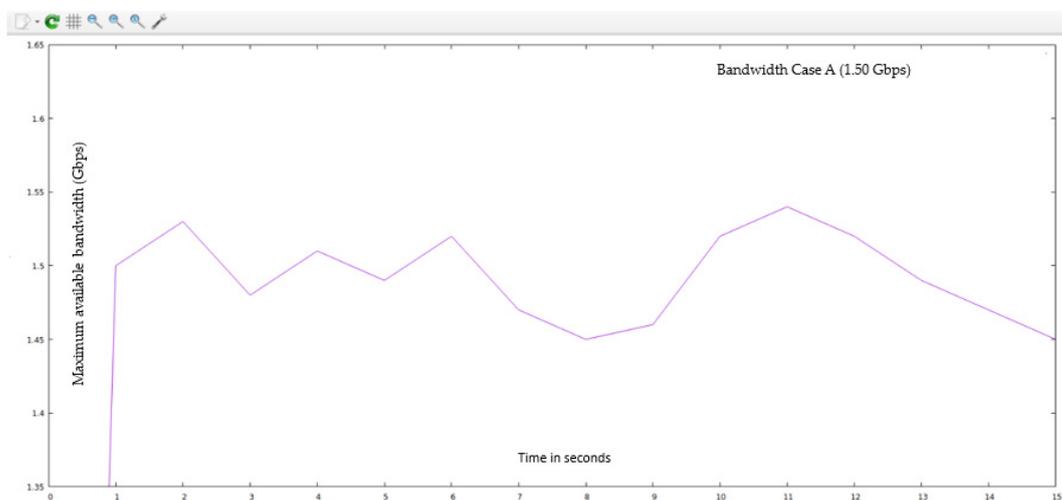
The “15,000” HTTP requests were generated from the randomly available twenty-seven hosts (user-defined network topology is shown in Figure 8) and forwarded to the POX controller. In this case, we have not defined the server load range of 1000 HTTP requests per second but instead measured the response time of HTTP servers in a user-defined network (as shown in Figure 8), as suggested in the research article [70]. This task is accomplished by sending an ARP packet to the servers. The server responds to the controller with less latency, and the new HTTP request flow is forwarded to that server. The maximum available bandwidth and transfer rate was calculated using the I-Perf utility for 15 s. The statistical data (QoS parameters) are shown in Figure 18a. Figure 18b,c represent the bandwidth and transfer rate of the test Case A (with the implementation of the research technique mentioned in the article [70] using the least server response method).

```

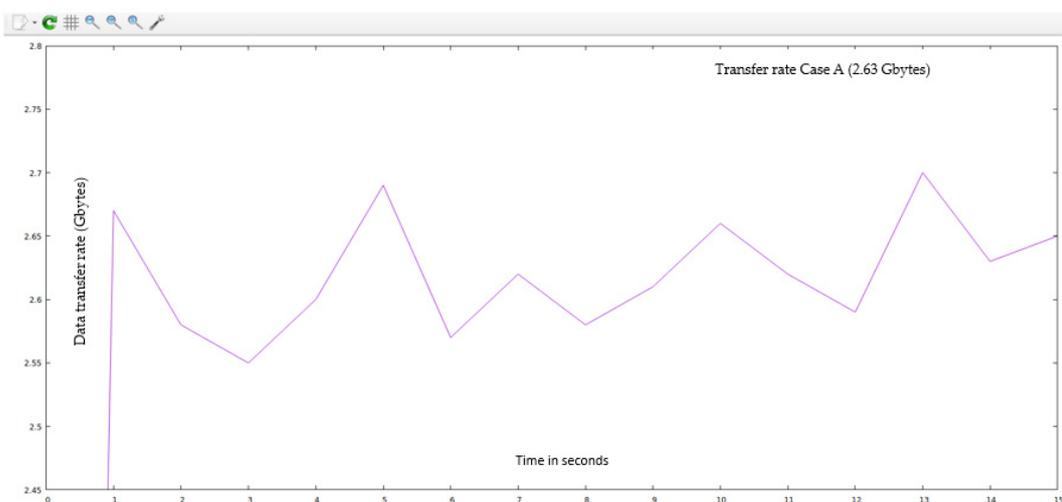
root@mininet-topology:~# iperf -c 10.0.1.1 -p 5566 -t 15
-----
Client connecting to 10.0.1.1, TCP port 5566
TCP window size: 85.3 KByte (default)
-----
[ 3] local 10.0.1.2 port 34580 connected with 10.0.1.1 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-15.0 sec  2.63 GBytes 1.50 Gbits/sec
root@mininet-topology:~# iperf -c 10.0.1.1 -p 5566 -t 15

```

(a)



(b)



(c)

Figure 18. (a) Summary of statistical data obtain from Iperf utility for test Case A. (b) QoS parameter (maximum available bandwidth) for test Case A (using the least server response method). (c) QoS parameter (transfer rate) for test Case A (using the least server response method).

Case B (using the traditional Round-Robin method):

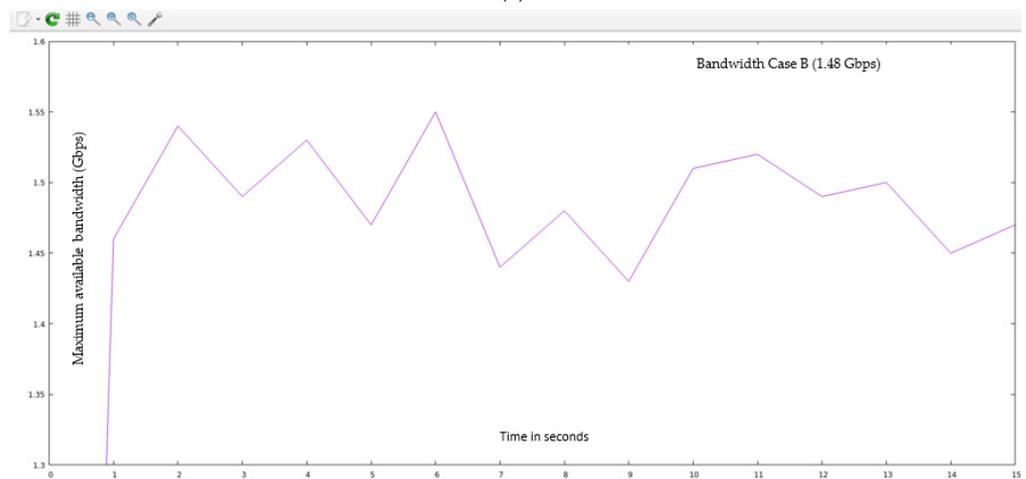
The “15,000” HTTP requests were generated from the randomly available twenty-seven hosts (user-defined network topology is shown in Figure 8) and forwarded to the POX controller. In this case, we have not defined the server load range of 1000 HTTP requests per second but instead applied the traditional Round-Robin approach in a user-defined network (as shown in Figure 8), as suggested in the research article [71]. The I-Perf utility is used to find the B_{am} and T_f . The statistical data (QoS parameters) are shown in Figure 19a.

```

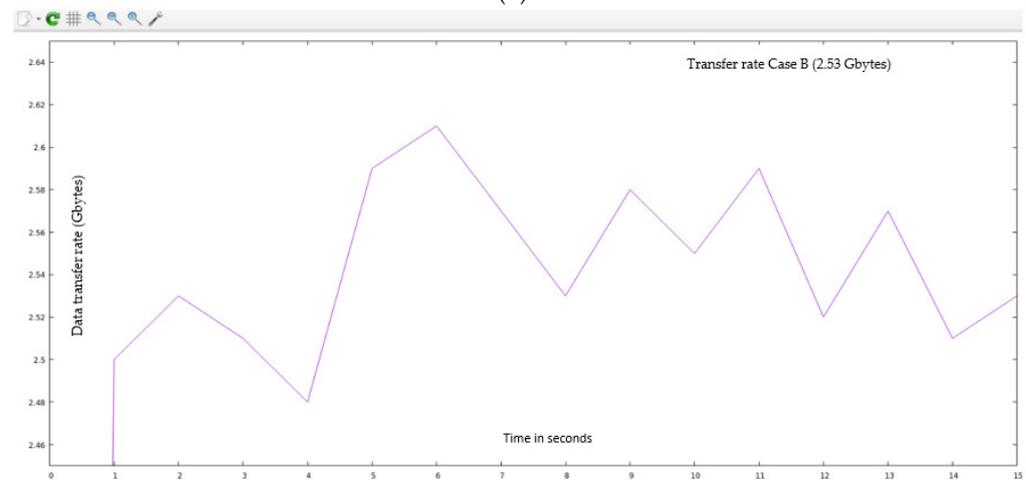
root@mininet-topology:~# iperf -c 10.0.1.1 -p 5566 -t 15
-----
Client connecting to 10.0.1.1, TCP port 5566
TCP window size: 85.3 KByte (default)
-----
[ 3] local 10.0.1.2 port 36456 connected with 10.0.1.1 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 3]  0.0-15.0 sec  2.59 GBytes 1.48 Gbits/sec
root@mininet-topology:~#

```

(a)



(b)



(c)

Figure 19. (a) Summary of statistical data obtain from Iperf utility for test Case B. (b) QoS parameter (maximum available bandwidth) for test Case B (using the traditional Round-Robin method). (c) QoS parameter (transfer rate) for test Case B (using the traditional Round-Robin method).

Figure 19b,c represent test Case B’s bandwidth and transfer rate (with the implementation of the research article [71] using the traditional Round-Robin method).

A comparison of QoS results obtained from Case A and B with the proposed algorithm (DASLM):

Table 8 summarizes the QoS parameters obtained in Cases A and B with the proposed algorithm (DASLM).

Table 8. Comparison of QoS obtained from I-Perf utility.

Method Used	Time	B _{am}	T _h	T _f (in 15 s)
QoS parameters with DASLM Algorithm.	0–15 s	3.48 Gbps	3.23 Gbps	6.07 Gbytes
Case B	0–15 s	1.48 Gbps	1.38 Gbps	2.59 Gbytes
Case A	0–15 s	1.50 Gbps	1.40 Gbps	2.63 Gbytes

Figure 20 illustrates the comparative analysis of QoS parameters (maximum available bandwidth and transfer rate).

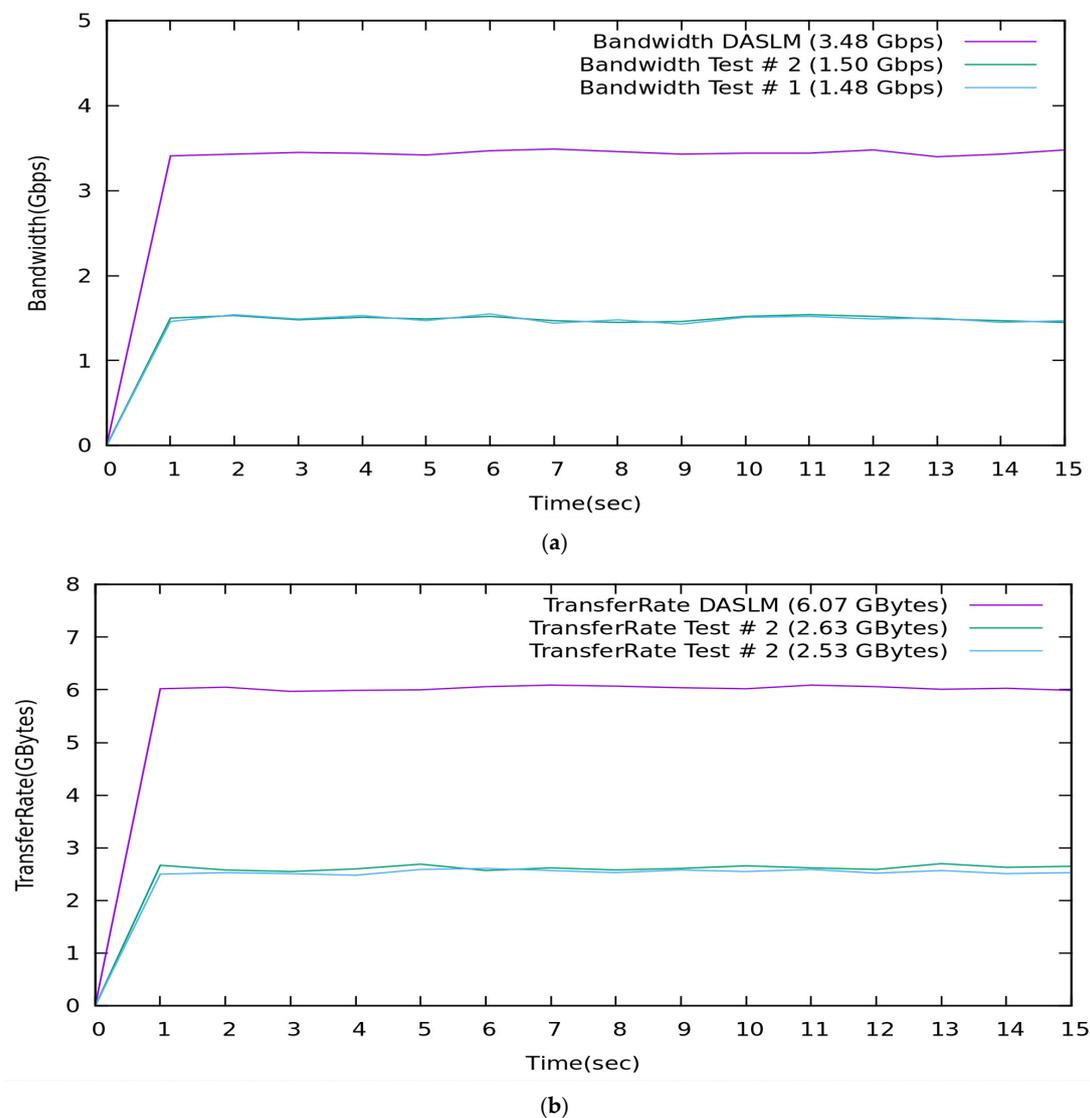


Figure 20. (a) Comparison of DASLM with traditional load-balancing method regarding QoS parameter (maximum available bandwidth). (b) Comparison of DASLM with traditional load-balancing method regarding QoS parameter (transfer rate).

A Summary of the Comparative Analysis:

With reference to Table 8, the QoS parameters verify that the maximum available bandwidth, throughput, and transfer rate of a user-defined network are improved by implementing the proposed technique in a real-time controlled SDN environment. The performance of the proposed algorithm (DASLM) is far superior to the method mentioned in the research articles [70,71].

5. Conclusions

In this study, we achieved the desired results by enhancing the QoS parameters of the HTTP server-based telecom network with the implementation of the proposed server load-balancing technique in a real-time controlled SDN environment, which is the DASLM (dynamic active sensing load managing) algorithm. The simulation in this manuscript was performed in two parts for 15 s: (1) QoS parameter analysis without implementing the DASLM algorithm; (2) QoS parameter analysis with the implementation of the DASLM algorithm. QoS analyses using I-Perf and Gnu-plot utility in the above-mentioned cases were performed based on two scenarios (normal flow with 150 HTTP requests and loaded flow with “15,000” HTTP requests). The QoS parameters in normal flow (with negligible load on the network servers) are considered a reference value to determine the network’s performance in loaded conditions (“15,000” HTTP requests in our simulation model). With the implementation of the proposed technique (DASLM), the QoS parameters (B_{am} , T_f , T_h , and L) have increased from 943 Mbps to 3.48 Gbps, 1.65 Gbytes to 6.07 Gbytes, 0.88 Gbps to 3.23 Gbps, and 12 ms to 0.87 ms, respectively, under loaded conditions (“15,000” HTTP requests). The maximum available bandwidth percentage ($\%B_{max}$) has increased from 4.78% (without DASLM implementation in the loaded scenario) to 86.57% (with DASLM implementation in the loaded scenario). The achievable transfer rate percentage ($\%T_{af}$) has also increased from 4.65% (without DASLM implementation in the loaded scenario) to 86.47% (with DASLM implementation in the loaded scenario). These QoS parameters verify that the maximum available bandwidth, throughput, and transfer rate are improved by the implementation of the proposed method (DASLM). For the authenticity of the proposed algorithm, the QoS results obtained from DASLM were compared with the QoS results obtained from the traditional server load-balancing algorithm: (a) server load balancing by calculating the least server response time method and (b) server load balancing by the traditional Round-Robin method; however, the values of the QoS parameters (B_{am} , T_f , T_h , and L) in the proposed algorithm (DASLM) were far superior to the traditional load-balancing methods and prove the effectiveness of the proposed technique. For future work, the proposed algorithm can be applied to the hierarchical, logically distributed SDN controller environment for server load management, where the whole network is divided into local domains. Each domain will have its controller work under the control parameters of the route and universal controller.

Author Contributions: The authors contributions to this paper are as follows: study conception and design, K.T.M. and S.A.; data collection, K.T.M. and S.A.; analysis and interpretation of results, K.T.M. and S.A.; draft manuscript preparation, K.T.M., S.A. and M.M.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are available from the corresponding author and can be provided upon appropriate request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Riyaz, M.; Musa, S. A Systematic Review of Load Balancing Technique in Software-Defined Networking. *IEEE Access* **2020**, *8*, 98612–98636.
2. Liu, C.; Ju, W. An SDN-Based Active Measurement Method to Traffic QoS Sensing for Smart Network Access. *Wirel. Netw.* **2021**, *27*, 3677–3688. [[CrossRef](#)]
3. Villalba, L.J.G.; Bi, J.; Jayasumana, A.P. Advances on Software Defined Sensor, Mobiles and Fixed Networks. *Int. J. Distrib. Sens. Netw.* **2016**, *12*, 5153718. [[CrossRef](#)]
4. Ghalwash, H.; Huang, C. Software Defines Extreme Scale Networking for Big Data Applications. In Proceedings of the High-Performance Extreme Computing Conference (HPEC'17), Waltham, MA, USA, 12–14 September 2017.
5. Andrus, B.; Olmos, J.J.V. SDN Data Center Performance Evaluation of Torus & Hypercube Inter-Connecting Schemes. In Proceedings of the Advances in Wireless and Optical Communication, Riga, Latvia, 5–6 November 2015; pp. 110–112.
6. Gau, R.-H. Optimal Traffic Engineering and Placement of VM in SDN with Service Chaining. In Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft), Bologna, Italy, 3–7 July 2017.
7. Schwabe, A.; Rajos, E. Minimizing Downtime: Using Dynamic Reconfigurations and State Management in SDN. In Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft), Bologna, Italy, 3–7 July 2017.
8. Lin, C.; Wang, K. A QoS Aware Routing in SDN Hybrid Network. *Procedia Comput. Sci.* **2017**, *110*, 242–249. [[CrossRef](#)]
9. Pasquini, R.; Stadler, R. Learning End-to-End Applications QoS from Open Flow Switch Statistics. In Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft), Bologna, Italy, 3–7 July 2017.
10. Zhang, T.; Liu, B. Exposing End-to-End Delay in SDN. *Hindawi Int. J. Reconfigurable Comput.* **2019**, *2019*, 7363901.
11. Ghalwash, H.; Huang, C.-H. A QoS framework for SDN-Based Networking. In Proceedings of the 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC), Philadelphia, PA, USA, 18–20 October 2018.
12. Shu, Z.; Wan, J. Traffic Engineering in SDN: Measurement and Management. In *IEEE Special Section on Green Communication and Networking for 5G Wireless*; IEEE: Piscataway, NJ, USA, 2016; p. 6258274.
13. Aly, W.H.F. Generic Controller Adaptive Load Balancing (GCALB) for SDN Networks. *J. Comput. Netw. Commun.* **2019**, *2019*, 6808693. [[CrossRef](#)]
14. Xu, Y.; Cello, M.; Wang, I.-C.; Walid, A.; Wilfong, G.; Wen, C.H.-P.; Marchese, M.; Chao, H.J. Dynamic switch migration in distributed software-defined networks to achieve controller load balance. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 515–529. [[CrossRef](#)]
15. Gao, Y.; Zhang, Z.; Zhao, D.; Zhang, Y.; Luo, T. A hierarchical routing scheme with load balancing in software defined vehicular ad hoc networks. *IEEE Access* **2018**, *6*, 73774–73785. [[CrossRef](#)]
16. Vyakaranal, S.B.; Naragund, J.G. Weighted round-robin load balancing algorithm for software-defined network. In *Emerging Research in Electronics, Computer Science and Technology*; Springer: Singapore, 2019; pp. 375–387.
17. Tu, R.; Wang, X.; Zhao, J.; Yang, Y.; Shi, L.; Wolf, T. Design of a load-balancing middlebox based on SDN for data centers. In Proceedings of the 2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Hong Kong, China, 26 April–1 May 2015; pp. 480–485.
18. Hu, Y.; Luo, T.; Wang, W.; Deng, C. On the load balanced controller placement problem in software-defined networks. In Proceedings of the 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, China, 14–17 October 2016; pp. 2430–2434.
19. Zhao, Y.; Liu, C.; Wang, H.; Fu, X.; Shao, Q.; Zhang, J. Load balancing-based multi-controller coordinated deployment strategy in software-defined optical networks. *Opt. Fiber Technol.* **2018**, *46*, 198–204. [[CrossRef](#)]
20. Jiugen, S.; Wei, Z.; Kunying, J.; Ying, X. Multi-controller deployment algorithm based on load balance in a software-defined network. *J. Electron. Inf. Technol.* **2018**, *40*, 455–461.
21. Wang, Q.; Gao, L.; Yang, Y.; Zhao, J.; Dou, T.; Fang, H. A Load-Balanced Algorithm for Multi-Controller Placement in a Software-Defined Network. *Mech. Syst. Control Formerly Control Intell. Syst.* **2018**, *46*, 72–81. [[CrossRef](#)]
22. Ma, Y.-W.; Chen, J.-L.; Tsai, Y.-H.; Cheng, K.-H.; Hung, W.-C. Load-Balancing Multiple Controllers Mechanism for Software-Defined Networking. *Wireless Pers. Commun.* **2017**, *94*, 3549–3574. [[CrossRef](#)]
23. Kaur, S.; Kumar, K.; Singh, J.; Ghuman, N.S. Round-Robin Based Load Balancing in Software-Defined Networking. In Proceedings of the 2nd International Conference on Computing for Sustainable Global Development At: Bharati Vidyapeeth's Institute of Computer Applications and Management (BVICAM), New Delhi, India, 11–13 March 2015; pp. 2136–2139.
24. Mulla, M.M.; Raikar, M.; Meghana, M.; Shetti, N.S.; Madhu, R. Load Balancing for Software-Defined Networks. In *Emerging Research in Electronics, Computer Science and Technology*; Springer: Singapore, 2019; pp. 235–244.
25. Chenhao, G.; Hengyang, W. An Improved Dynamic Smooth Weighted Round-robin Load-balancing Algorithm. In *Journal of Physics: Conference Series, Proceedings of the 2nd International Conference on Electrical Engineering and Computer Technology (ICEECT 2022), Suzhou, China, 23–25 September 2022*; IOP Publishing: Bristol, UK, 2022; Volume 2404, p. 2404.
26. Al-Tam, F.; Correia, N. On Load Balancing via Switch Migration in Software-Defined Networking. *IEEE Access* **2019**, *7*, 95998–96010. [[CrossRef](#)]
27. Linn, A.S.; Win, S.H.; Win, S.T. Server Load Balancing in Software Defined Networking. *Natl. J. Parallel Soft Comput.* **2019**, *1*, 261–265.
28. Kaur, P.; Bhandari, A. A Comparison of Load Balancing Strategy in Software Defined Networking. *Int. J. Res. Electron. Comput. Eng. IJRECE* **2018**, *6*, 1018–1025.

29. Yang, H.; Pan, H.; Ma, L. A Review on Software Defined Content Delivery Network: A Novel Combination of CDN and SDN. *IEEE Access* **2023**, *11*, 43822–43843. [[CrossRef](#)]
30. Singh, I.T.; Singh, T.R.; Sina, T. Server Load Balancing with Round Robin Technique in SDN. In Proceedings of the 2022 International Conference on Decision Aid Sciences and Applications (DASA), Chiangrai, Thailand, 23–25 March 2022.
31. Shona, M.; Sharma, R. Implementation and Comparative Analysis of Static and Dynamic Load Balancing Algorithms in SDN. In Proceedings of the 2023 International Conference for Advancement in Technology (ICONAT), Goa, India, 24–26 January 2023; pp. 1–7.
32. Karnani, S.; Shakya, H.K. Leveraging SDN for Load Balancing on Campus Network (CN). In Proceedings of the 2021 13th International Conference on Computational Intelligence and Communication Networks (CICN), Lima, Peru, 22–23 September 2021; pp. 167–171.
33. Sharma, R.; Reddy, H. Effect of Load Balancer on Software-Defined Networking (SDN) based Cloud. In Proceedings of the 2019 IEEE 16th India Council International Conference (INDICON), Rajkot, India, 13–15 December 2019.
34. Huang, W.-Y.; Chou, T.-Y.; Hu, J.-W.; Liu, T.-L. Automatical End-to-End Topology Discovery and Flow Viewer on SDN. In Proceedings of the 2014 28th International Conference on Advanced Information Networking and Applications Workshops, Victoria, BC, Canada, 13–16 May 2014; pp. 910–915.
35. Pakzad, F.; Portmann, M.; Tan, W.L.; Indulska, J. Efficient Topology Discovery in Software-Defined Networks. In Proceedings of the 2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS), Gold Coast, QLD, Australia, 15–17 December 2014; pp. 1–8.
36. Yuan, L.; Chuah, C.N.; Mohapatra, P. ProgME: Towards Pro-Programmable Network Measurement. *IEEE/ACM Trans. Netw.* **2011**, *19*, 115–128. [[CrossRef](#)]
37. Tootoonchian, A.; Ghobadi, M.; Ganjali, Y. OpenTM: Traffic Matrix Estimator for OpenFlow Networks. In *Passive and Active Measurement*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 201–210.
38. Malboubi, M.; Wang, L.; Chuah, C.-N.; Sharma, P. Intelligent SDN Based Traffic (De) Aggregation and Measurement Paradigm (iSTAMP). In Proceedings of the IEEE INFOCOM 2014—IEEE Conference on Computer Communications, Toronto, ON, Canada, 27 April 2014–2 May 2014; pp. 934–942.
39. Hu, Z.; Luo, J. Cracking Network Monitoring in DCNs with SDN. In Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM), Hong Kong, China, 26 April 2015–1 May 2015; pp. 199–207.
40. van Adrichem, N.L.M.; Doerr, C.; Kuipers, F.A. OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks. In Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; pp. 1–8.
41. Yu, C.; Lumezanu, C.; Zhang, Y.; Singh, V.; Jiang, G.; Madhyastha, H.V. FlowSense: Monitoring Network Utilization with Zero Measurement Cost. In *PAM 2013: Passive and Active Measurement*; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7799, pp. 31–41.
42. Mehmood, K.T.; Ateeq, S.; Hashmi, M.W. Predictive Analysis of Telecom System Quality Parameters with SDN (Software Define Networking) Controlled Environment. *Ilkog. Online-Elem. Educ. Online* **2020**, *19*, 5562–5575.
43. NetFlow. 2004. Available online: <http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html> (accessed on 17 September 2004).
44. sFlow. 2004. Available online: <http://www.sflow.org/sFlowOverview.pdf> (accessed on 1 July 2004).
45. Myers, A.C. JFlow: Practical Mostly-Static Information Flow Control. In Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, 20–22 January 1999; pp. 228–241.
46. Chowdhury, S.; Bari, M.F.; Ahmed, R.; Boutaba, R. PayLess: A Low-Cost Network Monitoring Framework for Software-Defined Networks. In Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; pp. 1–9.
47. Yu, M.; Jose, L.; Miao, R. Software Defined Traffic Measurement with OpenSketch. In Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), Lombard, IL, USA, 2–5 April 2013; pp. 29–42.
48. Moshref, M.; Yu, M.; Govindan, R.; Vahdat, A. DREAM: Dynamic Resource Allocation for Software-Defined Measurement. In Proceedings of the SIGCOMM '14: Proceedings of the 2014 ACM Conference on SIGCOMM, Chicago, IL, USA, 17–22 August 2014.
49. Mehdi, S.A.; Khalid, J.; Khayam, S.A. Revisiting Traffic Anomaly Detection Using Software-Defined Networking. In Proceedings of the RAID 2011: Recent Advances in Intrusion Detection, Menlo Park, CA, USA, 20–21 September 2011; pp. 161–180.
50. Zuo, Q.; Chen, M.; Wang, X.; Liu, B. Online Traffic Anomaly Detection Method for SDN. *J. Xidian Univ.* **2015**, *42*, 155–160. (In Chinese)
51. Canini, M.; Venzano, D.; Perešini, P.; Kostić, D.; Rexford, J. A NICE Way to Test OpenFlow Applications. In Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), San Jose, CA, USA, 25–27 April 2012; pp. 127–140.
52. Khurshid, A.; Zou, X.; Zhou, W.; Caesar, M.; Godfrey, P. VeriFlow: Verifying Network-Wide Invariants in Real Time. In Proceedings of the SIGCOMM '12: ACM SIGCOMM 2012 Conference, Helsinki, Finland, 13 August 2012; pp. 15–27.
53. Dixit, A.; Prakash, P.; Hu, Y.; Kompella, R. On the Impact of Packet Spraying in Data Center Networks. In Proceedings of the 2013 Proceedings IEEE INFOCOM, Turin, Italy, 14–19 April 2013; pp. 2130–2138.

54. Chiesa, M.; Kindler, G.; Schapira, M. Traffic Engineering with Equal-Cost-Multipath: An Algorithmic Perspective. In Proceedings of the IEEE INFOCOM 2014—IEEE Conference on Computer Communications, Toronto, ON, Canada, 27 April–2 May 2014; pp. 1590–1598.
55. Han, G.; Dong, Y.; Guo, H.; Shu, L.; Wu, D. Cross-Layer Optimized Routing in Wireless Sensor Networks with Duty Cycle and Energy Harvesting. *Wireless Commun. Mobile Comput.* **2015**, *15*, 1957–1981. [[CrossRef](#)]
56. Chen, M.; Leung, V.C.M.; Mao, S.; Yuan, Y. Directional Geographical Routing for Real-Time Video Communications in Wireless Sensor Networks. *Comput. Commun.* **2007**, *30*, 3368–3383. [[CrossRef](#)]
57. Chen, M.; Leung, V.C.M.; Mao, S.; Li, M. Cross-Layer and Path Priority Scheduling Based Real-Time Video Communications over Wireless Sensor Networks. In Proceedings of the VTC Spring 2008—IEEE Vehicular Technology Conference, Singapore, 11–14 May 2008; pp. 2873–2877.
58. Al-Fares, M.; Radhakrishnan, S.; Raghavan, B.; Huang, N.; Vahdat, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. In Proceedings of the NSDI'10: The 7th USENIX Conference on Networked Systems Design and Implementation, San Jose, CA, USA, 28–30 April 2010; Volume 10, p. 19.
59. Curtis, A.R.; Kim, W.; Yalagandula, P. Mahout: Low-Overhead Data-Center Traffic Management Using End-Host-Based Elephant Detection. In Proceedings of the 2011 Proceedings IEEE INFOCOM, Shanghai, China, 10–15 April 2011; pp. 1629–1637.
60. Benson, T.; Anand, A.; Akella, A.; Zhang, M. MicroTE: Fine-Grained Traffic Engineering for Data Centers. In Proceedings of the Co-NEXT '11: Conference on emerging Networking Experiments and Technologies, Tokyo, Japan, 6–9 December 2011; pp. 1–12.
61. Curtis, A.R.; Mogul, J.C.; Tourrilhes, J.; Yalagandula, P.; Sharma, P.; Banerjee, S. DevoFlow: Scaling Flow Management for High-Performance Networks. *ACM SIGCOMM Comput. Commun. Rev.* **2011**, *41*, 254–265. [[CrossRef](#)]
62. Yu, M.; Rexford, J.; Freedman, M.J.; Wang, J. Scalable Flow-Based Networking with DIFANE. *SIGCOMM Comput. Commun. Rev.* **2010**, *40*, 351–362. [[CrossRef](#)]
63. Silva, T.; Arsenio, A. A Survey on Energy Efficiency for the Future Internet. *Int. J. Comput. Commun. Eng.* **2013**, *2*, 595–689. [[CrossRef](#)]
64. Chen, M.; Zhang, Y.; Li, Y.; Mao, S.; Leung, V.C.M. EMC: Emotion-Aware Mobile Cloud Computing in 5G. *IEEE Netw.* **2015**, *29*, 32–38. [[CrossRef](#)]
65. Ke, B.-Y.; Tien, P.-L.; Hsiao, Y.-L. Parallel Prioritized Flow Scheduling for Software-Defined Data Center Network. In Proceedings of the 2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR), Taipei, Taiwan, 8–11 July 2013; pp. 217–218.
66. Li, D.; Shang, Y.; Chen, C. Software Defined Green Data Center Network with Exclusive Routing. In Proceedings of the IEEE INFOCOM 2014—IEEE Conference on Computer Communications, Toronto, ON, Canada, 27 April–2 May 2014; pp. 1743–1751.
67. Amokrane, A.; Langar, R.; Boutabayz, R.; Pujolle, G. Online Flow-Based Energy Efficient Management in Wireless Mesh Networks. In Proceedings of the 2013 IEEE Global Communications Conference (GLOBECOM), Atlanta, GA, USA, 9–13 December 2013; pp. 329–335.
68. Ali, S.; Alivi, M.K. Detecting DDoS Attack on SDN due to Vulnerabilities in OpenFlow. In Proceedings of the 2019 International Conference on Advances in the Emerging Computing Technologies (AECT), Al Madinah Al Munawwarah, Saudi Arabia, 10 February 2020.
69. Sathyanarayana, S.; Moh, M. Joint route-server load balancing in software defined networks using ant colony optimization. In Proceedings of the 2016 International Conference on High Performance Computing & Simulation (HPCS), Innsbruck, Austria, 18–22 July 2016; pp. 156–163.
70. Zhong, H.; Fang, Y.; Cui, J. Reprint of 'LBBSRT': "An efficient SDN load balancing scheme based on server response time". *Future Gener. Comput. Syst.* **2018**, *80*, 409–416. [[CrossRef](#)]
71. Hamed, M.; Elhalawany, B.; Fouda, M. Performance analysis of applying load balancing strategies on different SDN environments. *Benha J. Appl. Sci. (BJSAS)* **2017**, *2*, 91–97. [[CrossRef](#)]
72. Arahunashi, A.; Vaidya, G.; Neethu, S.; Reddy, K.V. Implementation of Server Load Balancing Techniques Using Software-Defined Networking. In Proceedings of the 2018, 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions(CSITSS), Bengaluru, India, 20–22 December 2018; pp. 87–90.
73. Kaur, S.; Singh, J. Implementation of server load balancing in software defined networking. In *Information Systems Design and Intelligent Applications*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 147–157.
74. Kavana, H.; Kavya, V.; Madhura, B.; Kamat, N. Load balancing using SDN methodology. *Int. J. Eng. Res. Technol.* **2018**, *7*, 206–208.
75. Hamed, M.; ElHalawany, B.M.; Fouda, M.M.; Eldien, A.S.T. A new approach for server-based load balancing using software-defined networking. In Proceedings of the 2017 Eighth International Conference on Intelligent Computing and Information Systems(ICICIS), Cairo, Egypt, 5–7 December 2017; pp. 30–35.
76. Ejaz, S.; Iqbal, Z.; Shah, P.A.; Bukhari, B.H.; Ali, A.; Aadil, F. Traffic load balancing using software defined networking (SDN) controller as virtualized network function. *IEEE Access* **2019**, *7*, 46646–46658. [[CrossRef](#)]
77. Gasmelseed, H.; Ramar, R. Traffic pattern-based load-balancing algorithm in software-defined network using distributed controllers. *Int. J. Commun. Syst.* **2019**, *32*, e3841. [[CrossRef](#)]
78. Hai, N.T.; Kim, D.-S. Efficient load balancing for multi-controller in SDN-based mission-critical networks. In Proceedings of the 2016 IEEE 14th International Conference on Industrial Informatics (INDIN), Poitiers, France, 19–21 July 2016; pp. 420–425.

79. Chiang, M.; Cheng, H.; Liu, H.; Chiang, C. SDN-based server clusters with dynamic load balancing and performance improvement. *Clust. Comput.* **2021**, *24*, 537–558. [[CrossRef](#)]
80. Zhang, H.; Guo, X. SDN-based load balancing strategy for server cluster. In Proceedings of the 2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems, Shenzhen, China, 27–29 November 2014.
81. Kreutz, D.; Ramos, F.M.V.; Verissimo, P. Towards secure and dependable software-defined networks. In Proceedings of the HotSDN '13: The Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, China, 16 August 2013; pp. 55–60.
82. Yeganeh, S.H.; Tootoonchian, A.; Ganjali, Y. On scalability of software-defined networking. *IEEE Commun. Mag.* **2013**, *51*, 136–141. [[CrossRef](#)]
83. Gude, N.; Koponen, T.; Pettit, J.; Pfaff, B.; Casado, M.; McKeown, N.; Shenker, S. NOX: Towards an operating system for networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 105–110. [[CrossRef](#)]
84. Sezer, S.; Scott-Hayward, S.; Chouhan, P.; Fraser, B.; Lake, D.; Finnegan, J.; Viljoen, N.; Miller, M.; Rao, N. Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Commun. Mag.* **2013**, *51*, 36–43. [[CrossRef](#)]
85. Karakus, M.; Durresi, A. A survey: Control plane scalability issues and approaches in software-defined networking (SDN). *Comput. Netw.* **2017**, *112*, 279–293. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.