Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2024.053903

ARTICLE

Examining the Quality Metrics of a Communication Network with Distributed Software-Defined Networking Architecture

Khawaja Tahir Mehmood[1,2,*], Shahid Atiq[1], Intisar Ali Sajjad[3], Muhammad Majid Hussain[4] and Malik M. Abdul Basit[2]

[1]Department of Electrical Engineering, Khwaja Fareed University of Engineering and Information Technology, Rahim Yar Khan, 64200, Pakistan

[2]Department of Telecommunication Systems, Bahauddin Zakariya University, Multan, 60000, Pakistan

[3]Department of Electrical Engineering, University of Engineering and Technology, Taxila, 47050, Pakistan

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

*Corresponding Author: Khawaja Tahir Mehmood. Email: ktahir@bzu.edu.pk

Received: 13 May 2024 Accepted: 26 July 2024 Published: 27 September 2024

ABSTRACT

Software-Defined Networking (SDN), with segregated data and control planes, provides faster data routing, stability, and enhanced quality metrics, such as throughput (T_h), maximum available bandwidth ($B_{d(max)}$), data transfer ($D_{Transfer}$), and reduction in end-to-end delay ($D_{(E-E)}$). This paper explores the critical work of deploying SDN in large-scale Data Center Networks (DCNs) to enhance its Quality of Service (QoS) parameters, using logically distributed control configurations. There is a noticeable increase in $Delay_{(E-E)}$ when adopting SDN with a unified (single) control structure in big DCNs to handle Hypertext Transfer Protocol (HTTP) requests causing a reduction in network quality parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, $D_{(E-E)}$, etc.). This article examines the network performance in terms of quality matrices (bandwidth, throughput, data transfer, etc.), by establishing a large–scale SDN-based virtual network in the Mininet environment. The SDN network is simulated in three stages: (1) An SDN network with unitary controller-POX to manage the data traffic flow of the network without the server load management algorithm. (2) An SDN network with only one controller to manage the data traffic flow of the network with a server load management algorithm. (3) Deployment of SDN in proposed control arrangement (logically distributed controlled framework) with multiple controllers managing data traffic flow under the proposed Intelligent Sensing Server Load Management (ISSLM) algorithm. As a result of this approach, the network quality parameters in large-scale networks are enhanced.

KEYWORDS

Software defined networking; quality of service; hypertext transfer protocol; data transfer rate; latency; maximum available bandwidth; server load management

1 Introduction

In today's advanced technological environment, internet service providers increasingly manage demands for ultra-high-speed connectivity across different remote networks. By the end of 2023, approximately 29.3 billion different remote networks should be connected via modern network resources [1]. Optimizing the performance of any network mostly involves increasing its throughput, ensuring that it can process large amounts of data concurrently, improving data transfer rates, and minimizing latency. Software-defined networking (SDN) can ease the optimization process with a unique centralized control arrangement for managing data traffic and the ability to overcome the challenges associated with traditional networks in terms of scaling, controllability, and reliability [2]. SDN architectural framework has a three-layered structure (1-application layer, 2-control layer, and 3-data layer). Using a northbound Application Program Interface (API), the programmable logic is easily transferred to the controller at the control layer from the application layer. SDN controller updates routing tables of various networking devices in the data plane and coordinates and manages the data flow over various topologies via the southbound interface. SDN is becoming a key component of modern network infrastructures because of its two programmable properties that greatly enhance network scalability: (1) The segregation of the data plane from the control plane, augmenting network controllability, as illustrated in Fig. 1. (2) The instructions can easily be forwarded to the SDN controller that can effectively process data traffic management over the whole network architecture [3], providing augmented controllability. Due to centralized control arrangements, congestion in SDN-based networks can easily be managed compared to traditional networks [4]. In an SDN-based network, the segregation of control and data planes means all the flow management functionality of network devices (routers, switches, etc.) is managed by the SDN controller. The unified controller is used for small to medium-sized networks and multiple controllers (logically distributed/logically centralized). This functionality enhances the Quality of Service (QoS) parameters; the logical reasons are as follows:

- ➢ The separation of control and data planes with all flow management control is with an SDN controller, which can have dynamic flow control ability in an SDN-based network by managing network traffic flow based on real-time conditions.
- ➢ The separation of planes with the controller incharge provides reconfiguration of network paths, avoidance of congestion, application of prioritization of network traffic, enhanced throughput, greater data transfer, maximum available bandwidth, and ensuring low latency. These factors result in enhanced network performance and greater QoS parameter values.
- ➢ The segregation of control and data planes in an SDN-based network provides additional facilities to optimize resource allocation based on the current demands of users by modifying the controller operation mode.
- ➢ The programmability of the control plane allows quick implementation of new applications and services based on the QoS demands of any network environment.
- ➢ The segregation of data and control planes in SDN provides an agile, responsive, and dynamically manageable network, resulting in enhanced QoS parameters.

There are two configuration options for the SDN controller: (a) centralized mode and (b) distributed mode [5]. The SDN controller operates best in centralized mode for small- to medium-sized networks. However, there can be significant delays in data transmission when large, multi-component Data Center Networks (DCNs) use a single, centralized model-based SDN controller, which provides slower controller operation and causes traffic congestion and delay. This delay can exacerbate network issues by creating backlogs of data that need to be processed [6]. The study was conducted in [7] to evaluate the performance of a single NOX controller for processing a large amount of data to pass through a large DCN. To mitigate latency issues within large-scale SDN-based DCNs using a single controller, the SDN controller must be framed in either logically centralized or distributed [8,9] control mode. The research articles [10,11] suggest the resource management issue and its mitigation in the SDN network.

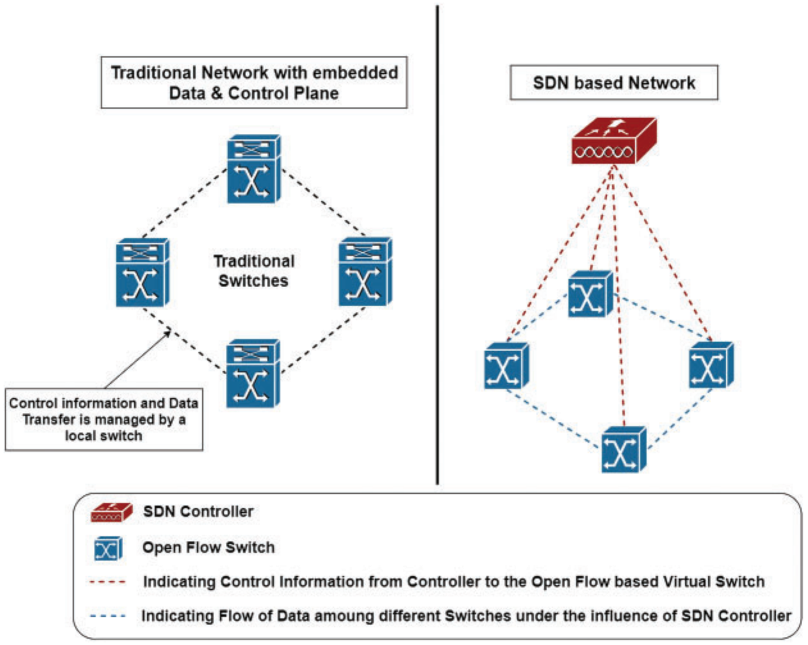

Traditional Network with embedded Data & Control Plane
Traditional Switches
Control information and Data Transfer is managed by a local switch
SDN based Network
SDN Controller
Open Flow Switch
Indicating Control Information from Controller to the Open Flow based Virtual Switch
Indicating Flow of Data amoung different Switches under the influence of SDN Controller

Figure 1: Difference between the traditional and SDN-based networks' operational models

1.1 A Centralized Controlled System Using an SDN Controller

This controller design distributes Hypertext Transfer Protocol (HTTP) requests from different hosts equally by deploying numerous high-performance SDN controllers across DCNs with exact synchronization. In comparison to the single SDN controller mode, this logically centralized SDN controller configuration, as shown in Fig. 2, improves QoS characteristics in bigger DCNs. On the other hand, maintaining strict synchronization between these controllers may cause compatibility issues. The arrangement's primary drawback is the difficulty of synchronization between various controllers. Under some circumstances, synchronizing these controllers with un-versioned third-party programs may cause unpredictable and unstable network behavior, ultimately leading to system failure.

1.2 Distributed Controlled Arrangement with an SDN Controller

This SDN control architecture uses a decentralized method that eliminates the need for controller synchronization in a distributed network configuration. The vast network of Large DCNs is separated into smaller domains, in contrast to the use of separate controllers in master and slave versions, as explained in Section 1.1. Every controller stores vital flow information inside its Storage Area Network (SAN) and controls flow within its assigned domain. The network administrator can use this data to receive more instruction, which improves localized flow control inside each domain. Due to this solution, the strict synchronization requirement of distributed controllers is removed, which is necessary for global control of the distributed SDN network, as outlined in Section 1.1. Comparing this SDN controller configuration to a centralized SDN control framework, one can get higher efficiency, faster response rates, and more versatility. Fig. 3 outlines the structure for logically distributed SDN control mode.

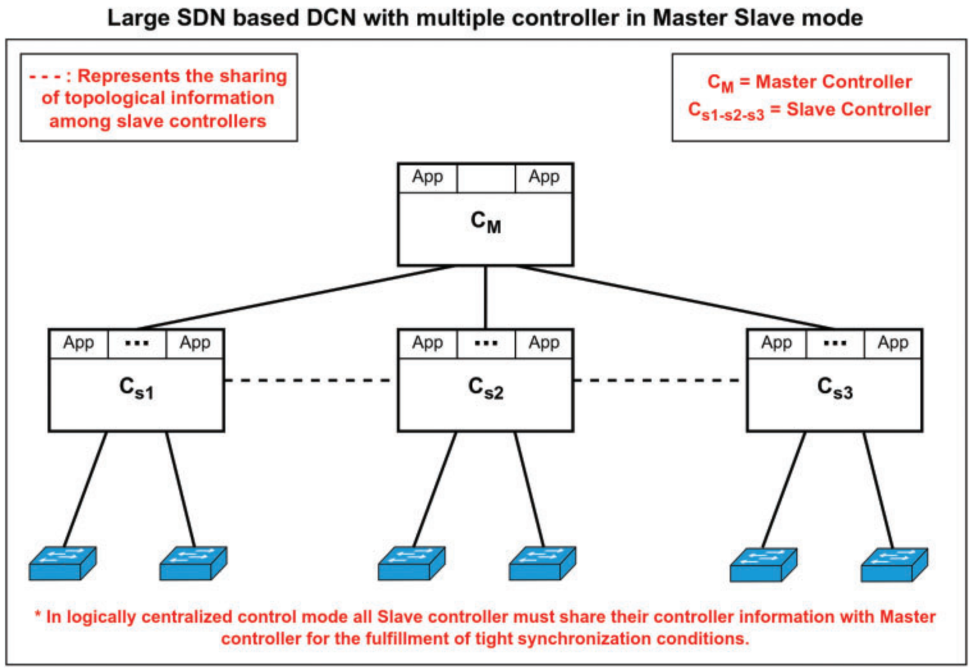

Large SDN based DCN with multiple controller in Master Slave mode
- - - : Represents the sharing of topological information among slave controllers
C_M = Master Controller
$C_{s1-s2-s3}$ = Slave Controller
App
App
C_M
App ••• App
C_{s1}
C_{s2}
C_{s3}
* In logically centralized control mode all Slave controller must share their controller information with Master controller for the fulfillment of tight synchronization conditions.

Figure 2: Logically centralized SDN controller configuration

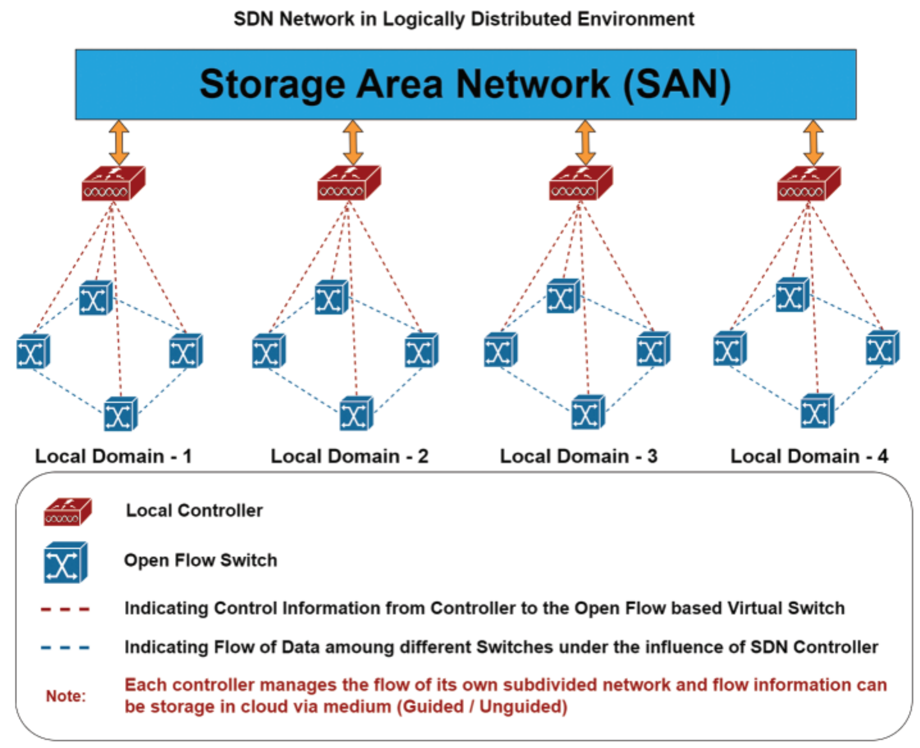

SDN Network in Logically Distributed Environment
Storage Area Network (SAN)
Local Domain - 1
Local Domain - 2
Local Domain - 3
Local Domain - 4
Local Controller
Open Flow Switch
Indicating Control Information from Controller to the Open Flow based Virtual Switch
Indicating Flow of Data amoung different Switches under the influence of SDN Controller
Note: Each controller manages the flow of its own subdivided network and flow information can be storage in cloud via medium (Guided / Unguided)

Figure 3: SDN controller set up in a control mode

This SDN control configuration is frequently used by software-defined wide-area networks (SD-WANs). The research articles [12,13] demonstrate how SDN is used in distributed controlled structures regarding Google and Microsoft's network development. These employ distributed network devices to offer their users services. Compared to typical networking systems, the quality characteristics with a hierarchical distributed control arrangement are significantly superior. Fig. 4 shows the hierarchical distributed control configuration. The controller of each domain in the network is in charge of it. Under the supervision of a global controller, each route controller controls its local controller. That local domain controller manages the data flow inside a domain. When data travels outside the local controller's controlling area, the route controller will take control of the flow. The research papers [14–16] raise attention to the issue with the "east-west" bound interface between multiple controllers.

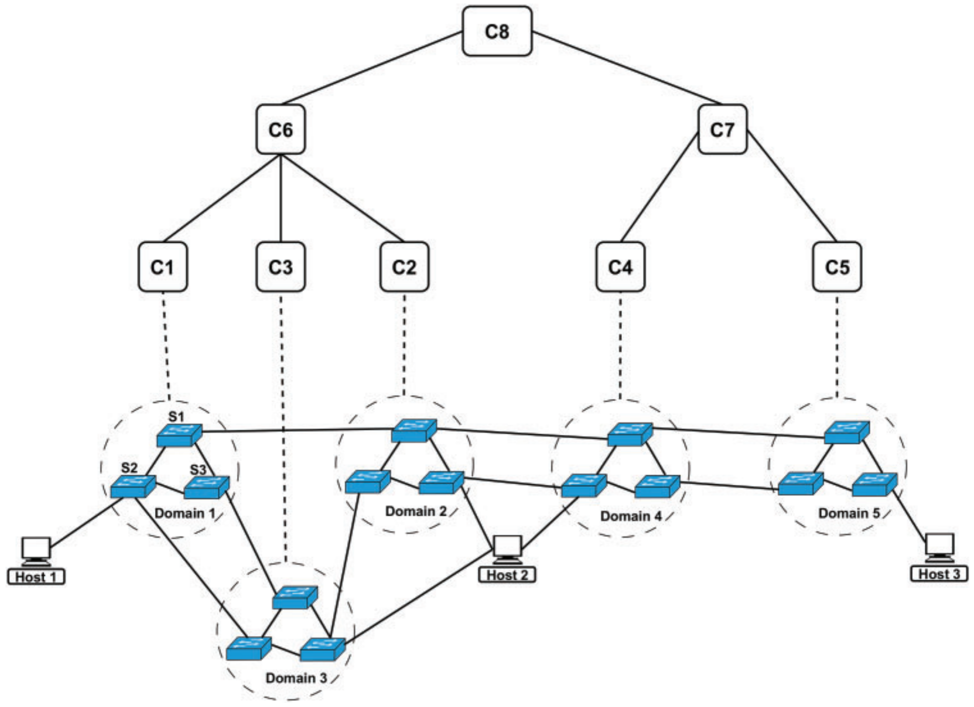


Figure 4: SDN controller arranged in hierarchical, logically distributed mode

1.3 Research Objectives

The main goal of this study is to implement the suggested method (ISSLM) to control network server loads effectively and improve the QoS. Many HTTP servers can handle HTTP queries within the SDN architecture. However, server overload may occur if all HTTP request flows are directed toward a single server without considering network server load management. Consequently, this results in (a) overcrowded links, (b) HTTP queues, and (c) end-to-end delays. In the end, this decrease in the QoS characteristics of the network may cause the network to collapse due to total network failure. The following are the main objectives of this research project.

1.3.1 To Deploy SDN in Large DCNs

In this method, the large DCN network is divided into smaller domains, and every controller is responsible for flow control inside its domain and storing necessary flow data in its SAN for the network administrator to instruct the controller further to manage the flow inside that local domain better as compared to the logically centralized method.

1.3.2 To Enhance QoS (B_{dmax}, T_h, $D_{Transfer}$, etc.) Parameters

A. Self-Adapting Configuration:

The Request Per Second (RPS) load on each server is calculated by adaptation of the following procedure:

➢ Firstly, the APACHE program must be installed on every HTTP server.

➢ Then, the GRAFANA program is started on every server. PROMETHEUS is a data source, making collecting HTTP server data logs possible.

➢ Prom_QL queries are written to control the RPS matrices for every server, considering settings like RAM and default time.

➢ Installing and integrating the POX controller with the PSUTIL Python library function.

➢ Using PSUTIL, endpoints are created so that the POX controller may access all of the servers' RPS matrix values.

➢ These RPS matrices are used to determine the load on every HTTP server.

B. Adaptive Optimization:

To balance server loads, the ISSLM algorithm takes three factors into account: (1) Comparing each server's RPS values to a benchmark server load (S_{load}) measurement. When a server's load exceeds the reference server load (S_{load}) in an SDN network, it is considered overloaded and is removed from the server list. When this occurs, HTTP flows are diverted to the server with the lowest server load value among the servers accessible on the network, within the acceptable range of the reference server load (S_{load}) value. (2) HTTP flows are routed to the server in the SDN network with the fewest active connections if the criterion mentioned above is not satisfied. (3) The server with the fastest response time.

1.4 Work Contribution

The main goal of this study is to apply a suggested load balancing algorithm, called ISSLM, for network servers to improve the QoS metrics, which include maximum bandwidth (B_{dmax}), throughput, data transfer, and delay in a chosen or parameterized SDN based DCN. The suggested working model, which uses the ISSLM algorithm to accomplish the study goals, is shown in Fig. 5.

A large-scale SDN-based virtual network was created in the Mininet environment to study network quality factors to achieve the previously indicated goal. According to this research study, the chosen or parameterized network is simulated in three steps:

➢ In the first step, we configured an SDN network with only one controller—the POX controller—to manage the data flow over the network. The Mininet environment is used to set up this network configuration. In particular, this configuration does not include a server load management algorithm.

➢ In the second step, we keep the SDN network operational using a single controller (POX) set up in the Mininet environment, which controls data traffic flow within the network. We integrate a server load management algorithm into this design to efficiently balance the load among network servers.

➢ In the third case, we build a user-defined distributed network that several controllers oversee. The controllers are responsible for monitoring data flow inside their designated domains. Moreover, the suggested ISSLM algorithm influences how these controllers function. This strategic approach aims to improve the large-scale network's network quality metrics.

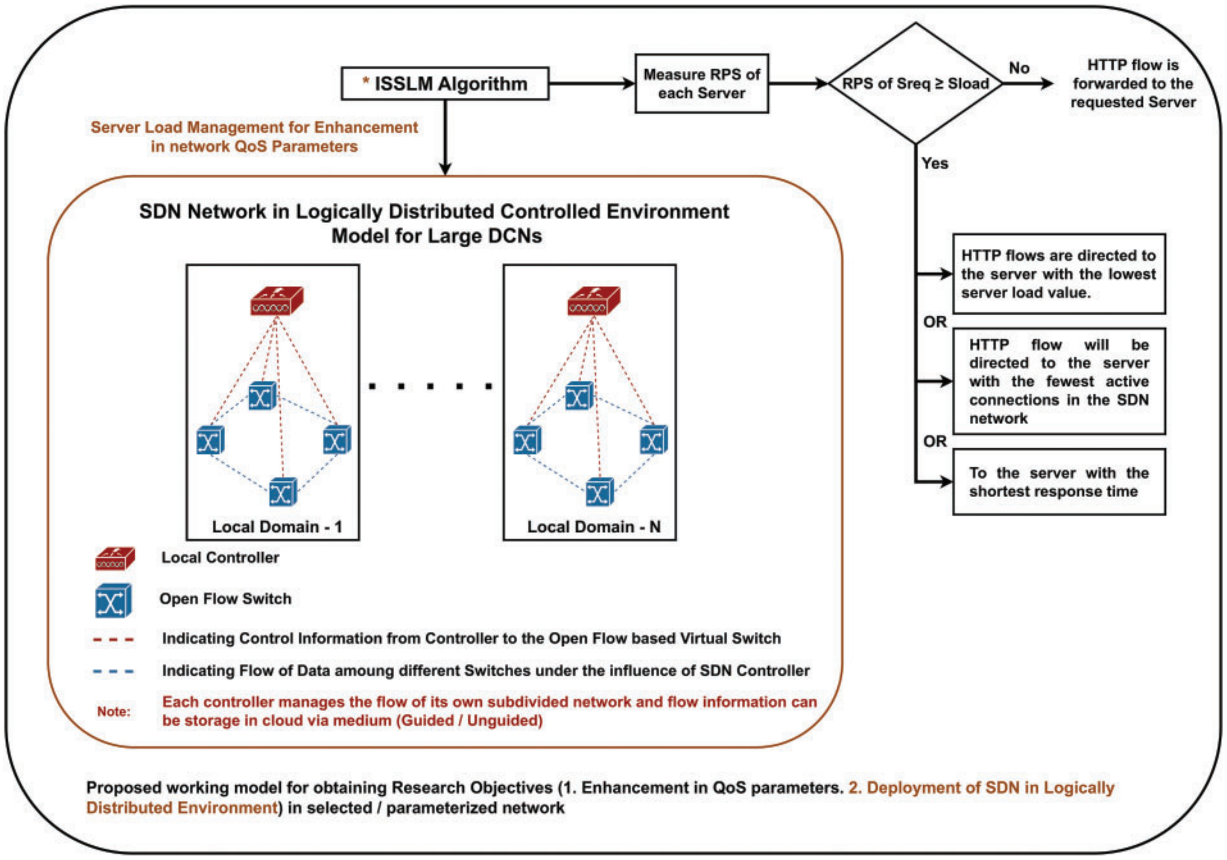

* ISSLM Algorithm
Measure RPS of each Server
RPS of Sreq ≥ Sload
No
HTTP flow is forwarded to the requested Server
Yes
Server Load Management for Enhancement in network QoS Parameters
SDN Network in Logically Distributed Controlled Environment Model for Large DCNs
Local Domain - 1
Local Domain - N
HTTP flows are directed to the server with the lowest server load value.
OR
HTTP flow will be directed to the server with the fewest active connections in the SDN network
OR
To the server with the shortest response time
Local Controller
Open Flow Switch
Indicating Control Information from Controller to the Open Flow based Virtual Switch
Indicating Flow of Data amoung different Switches under the influence of SDN Controller
Note:
Each controller manages the flow of its own subdivided network and flow information can be storage in cloud via medium (Guided / Unguided)
Proposed working model for obtaining Research Objectives (1. Enhancement in QoS parameters. 2. Deployment of SDN in Logically Distributed Environment) in selected / parameterized network

Figure 5: A proposed working model for obtaining research objectives by the proposed algorithm (ISSLM)

1.5 Paper Arrangement

Section 2 explains the literature review and provides a comparison table of traditional server load management algorithms with the proposed technique of the ISSLM algorithm. In Section 3, we have discussed the working model of (ISSLM) algorithm and problem statement. In Section 4, the SDN network (in this research article) is simulated in three stages: (1) An SDN network with only one controller (POX) to manage the data traffic flow of the network (established on the Mininet environment) without the server load management algorithm. (2) An SDN network with only one controller to manage the data traffic flow of the network (established on the Mininet environment) with a server load management algorithm. (3) A user-defined network is now established in distributed network domains with multiple controllers managing data traffic flow under the proposed ISSLM algorithm. As a result of this approach, the network quality parameters in the large-scale network are augmented. In Section 5, we conclude the simulation results along with future research directions.

2 Literature Review

This section provides an overview of study findings about distributed SDN control configurations divided into two groups. As shown in Fig. 6, there are two research strategies: (1) logically centralized SDN configuration research techniques and (2) logically distributed SDN configuration research techniques.

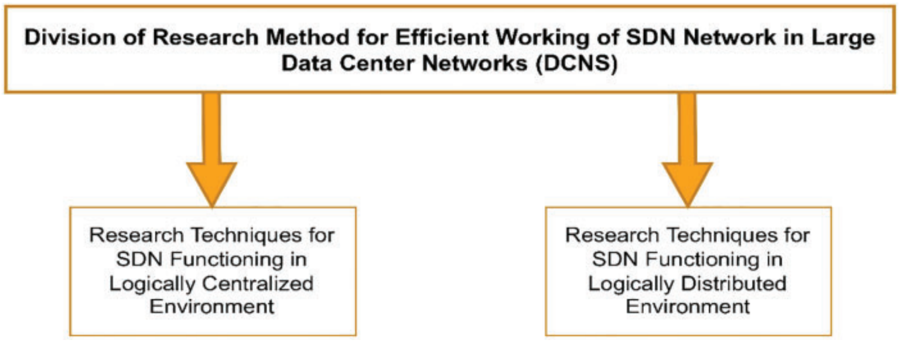

Division of Research Method for Efficient Working of SDN Network in Large Data Center Networks (DCNS)
Research Techniques for SDN Functioning in Logically Centralized Environment
Research Techniques for SDN Functioning in Logically Distributed Environment

Figure 6: Classification of the literature review into two portions

2.1 *Research Methodology for Linking Large DCNs in SDN Centralized Controlled Structure*

The Hyperflow protocol [17], which uses the WheelFS application to synchronize controllers, is covered in an article [18]. A broker-oriented distributed file system is used in this technique, which could lead to network Denial of Service (DOS) attacks. The Online Information Exchange (ONIX) program, which controls flow characteristics in the Network Information Database (NIDB) database, is presented in a research study [19]. While this method is appropriate for medium-sized networks, NIDB exchange may cause network overload in larger Data Center Networks (DCNs), resulting in worse quality parameters. A research paper [20] shows an Open Daylight (ODL) cluster that utilizes the AKKA (derived from a Swedish mountain and the choice of name symbolizes stability, resilience, and the ability to handle heavy loads, much like a mountain) framework [21,22] to synchronize controllers. Still, many control frames are used, making big DCNs slow. The Open Network Operating System (ONOS) is presented in a research article [23], which uses the Reliable, Available, and Fault-Tolerant (RAFT) algorithm [24] to map data between controllers. An active standby controller configuration is used to provide fault tolerance. According to a research paper [25], Atomix, a Java-based program, helps with fault tolerance in SDN networks. Similarly, Smart-Light uses an active-standby controller topology for fault tolerance with centralized data storage (CDS) [26,27]. A research article [28] discusses Ravenna, built on the Replicated State Machine (RSM) method for network device interconnections. As explained in another study [29], Manta's and Rama's methods aim to emulate Ravana's traits without changing the OpenFlow protocol. Compared to Ravana, the open flow-based algorithm Kandos provides better quality parameters, as shown in a study publication [30]. Furthermore, research papers [31,32] include white documents from the Google project, which show that when used in logically centralized control mode setups, practical efficiency can reach up to 90%.

2.2 *Research Methodology for Linking Large DCNs in SDN Distributed Controlled Structure*

The functioning of controllers in the SDN distributed controlled plane (DISCO) is explained in a research study [33]. For intra- and inter-domain applications, this controller uses a variety of network-based agents to enable information flow. A revised protocol for managing queue messages (AMQP), specifically for inter-domain data transfers, is suggested by the authors in a research study [34]. The outcomes are noticeably better with the application of this technique. The Distributed Control Plane Interface (DCPI) program, which functions as an east-west bound interface, is also introduced in a research study [35]. It makes sharing and synchronizing network flow status easier for several domains. Based on the OpenFlow protocol, the Route Flow Partial (RFP) application is used to manage applications related to inter-domain flow. It is covered in more detail in another article [36]. A research article [37] discusses an application created by INRIA to improve fault tolerance and security in distributed SDN (D-SDN) configurations. Novel techniques for implementing SDN across internet exchange points, or SDX, are explored in several research articles [38–42]. Among these, research articles [43–45] cover fault tolerance techniques and DOS attack detection techniques within SDX. Furthermore, a study [46] describes enhancements to the SDN network's Border Gateway Protocol (BGP). As demonstrated in a different study publication [47–48], the Distributed Flow Architecture for Networked Enterprises (DIFANE) application, when implemented in an SDN network, yields better QoS parameters.

2.3 A Comparison between Traditional Server Load Balancing Algorithms and the Proposed (ISSLM) Algorithm for Efficient Server Load Balancing

The differences between the suggested ISSLM algorithm and traditional load-balancing techniques are shown in Table 1. This section also details how the ISSLM algorithm overcomes the drawbacks of conventional load-balancing methods.

Table 1: Comparison of the proposed (ISSLM) method with conventional server load balancing algorithms

Authors	Key contribution	Implemented method/technique	Notable constraint
Mehmood et al. [49]	Improve the network's QoS metrics, focusing on $B_{d(max)}$, throughput, and data transfer.	Implementing an algorithm for dynamic load balancing.	Scalability problems in larger configurations; effectiveness restricted to smaller DCNs.
Chiang et al. [50]	1-Optimum servers are chosen based on lower RPS values. 2-Execution is carried out in a framework that is hardware-centric.	1-Utilizing the Dynamic Weighted Random Selection (DWRS) method. Giving less loaded servers a more substantial weight. 2-Direct flow to servers that have greater weights.	Lack of active connection analysis and rapid responses.
Begam et al. [51]	1-Best server selection based on calculation of response time. 2-Redirecting flow to servers with the lowest possible delay.	The search technique was based on multiple regression (MRBS).	The approach does not use active server load sensing.

(Continued)

Table 1 (continued)

Authors	Key contribution	Implemented method/technique	Notable constraint
Malbašić et al. [52]	1- Determine the optimal server based on minimum server load. 2-Implementation is performed on a Mininet arrangement.	Using matrices with several parameters to schedule connections.	Restricted scope of the assessments, absence of response time, and active connection assessments.
Liang et al. [53]	Automated load redistribution among servers driven by IoT applications.	Employing a Bayesian network based on IoT inputs.	Oversight in active server load sensing within the framework.
Ahmad et al. [54]	Middlebox integration as a proxy server for load sharing with SDN controllers.	Developing a robust network involving a controller and middlebox for traffic control.	Compatibility challenges between middlebox and SDN controller infrastructure.
Saxena et al. [55]	Introduction of Adaptive Multi-Objective Load Balancing (AMOLB) considering various metrics.	Calculation of routing cost with data forwarding based on path weights.	The narrow focus on routing cost overlooks response time considerations.
Haidi et al. [56]	Improving Bandwidth utilization through Dynamic Load Balancing.	The Floodlight controller is used for Dynamic Load Balancing.	Insufficient analysis of response time dynamics.
Ejaz et al. [57]	Load distribution among SDN controllers.	Multiple controllers in Master-Slave scenario.	Dependency on tight synchronization within logically centralized frameworks.
Gasmelseed et al. [58]	SDN controller load balancing.	TCP/UDP flow separation using Multiple Controllers.	Dependency on tight synchronization within logically centralized frameworks.
Xu et al. [59]	Load balancing on network switches.	It is based on migration cost, prioritizing servers with lower migration costs.	Inefficiency when targeting overloaded servers considering migration cost.

(Continued)

Table 1 (continued)

Authors	Key contribution	Implemented method/technique	Notable constraint
Geo et al. [60]	Bandwidth optimization & Latency reduction.	They are opting for multiple controllers with segregated master controllers.	Compatibility challenges, energy consumption, and overhead constraints with controllers.
Vyakarana et al. [61]	Elephant flow avoidance.	Employing a static algorithm distinguishing critical and non-critical traffic.	Network bandwidth wastage under varying dynamic load conditions.
Sathyanarayana et al. [62]	Load balancing via less-loaded server selection and shortest path algorithms.	1-Dynamic Flow Algorithm for servers with less traffic. 2-The shortest path to the best server using the Ant Colony Algorithm.	Extensive resource consumption when concurrently executing both algorithms.
Zhong et al. [63]	Reduction of processing delays.	Computations of server response time.	Trade-offs in throughput and bandwidth to mitigate processing delays.
Hamed et al. [64]	Server load balancing.	Utilizing the traditional Round-Robin method.	Reduced load balancing effectiveness in extensive SDN setups with intense data flows.
Hai et al. [65]	Elephant flow mitigation.	Using traffic categorization into critical and non-critical segments.	Significant network bandwidth utilization concerns.

3 Problem Statement

When a single SDN controller controls the data traffic flow of a large SDN-based DCN, even how efficient the proposed method is, the network quality parameters are derailed. In the research article [47], the server load was balanced using a dynamic server load algorithm but using a single controller, which could result in a large processing delay of control instruction to the underlying switch in large DCNs. This research article uses multiple controllers in a logically distributed scenario to overcome the tight synchronization problem, as happens in the logically centralized control environment. Many research ideas to address this practical issue are mentioned in Section 2. But there are some significant issues related to this worthwhile issue that are following:

➢ The SDN controller manages the data traffic flow of all networking devices inside a network. Suppose another controller is incorporated inside a network to share the load of flow control. The transfer of control information among controllers inside the network requires tight synchronization. The open flow protocol will connect them to transmit control information regarding network controllability, which could also lead to vulnerabilities in DOS attacks. The problem of synchronization arises in the logically controlled arrangement.

➢ However, suppose the synchronization condition is somehow fulfilled. In that case, there is a problem when the network is a large DCN, and sharing necessary control files among controllers in case of heavy data traffic flow could make the controller loaded, enhancing the processing delays and latency and further derailing network quality parameters.

➢ Normally, there are two types of data flow inside a network: (1) data traffic requests to be transferred from one host and (2) control data flow from one controller to another. Sharing every flow information among different controllers has the advantage that if one controller fails, another controller takes responsibility for flow control in addition to its commitments. This requires additional spaces and programming to sport multiple control data flow inside a network.

However, this research article proposed a more efficient method: to balance server load as compared to the research methods suggested in Table 1 that is, instead of using different controllers in the master and slave version, the large DCNS network is divided into smaller domains, and every controller is responsible for flow control inside its domain and storing necessary flow data in its SAN for the network administrator to instruct the controller further to manage the flow inside that local domain better.

Solution Proposed by Implementing the Proposed Algorithm

Logically distributed use of SDN controllers is recommended when working with big networks with hundreds of thousands of network components. This method divides the large DCNs into smaller domains and assigns each controller to control flow control inside its assigned domain. These controllers also store important flow data in their SANs, which makes it easier for network administrators to give additional instructions for better flow management in each local domain. As seen in Fig. 3, this logically distributed technique has advantages over the logically centralized approach.

4 Methodology of the Proposed Technique (ISSLM)

The methodology of the proposed algorithm, the ISSLM algorithm, is divided into three procedural steps that are as follows.

4.1 Procedural Step#1

The SDN controller mode of operation is defined before forming an SDN-based network. If the network to be managed is small (comprising a few networking devices), then a single SDN controller is sufficient to control the flow. However, if the network is large (comparing hundreds of thousands of network components), the SDN controller should be used in a logically distributed manner. In this method, the large DCNS network is divided into smaller domains, and every controller is responsible for flow control inside its domain and storing necessary flow data in its SAN for the network administrator to instruct the controller further to manage the flow inside that local domain better as compared to the method adopted in research article [49]. The procedure of Step 1 is shown in Fig. 7.

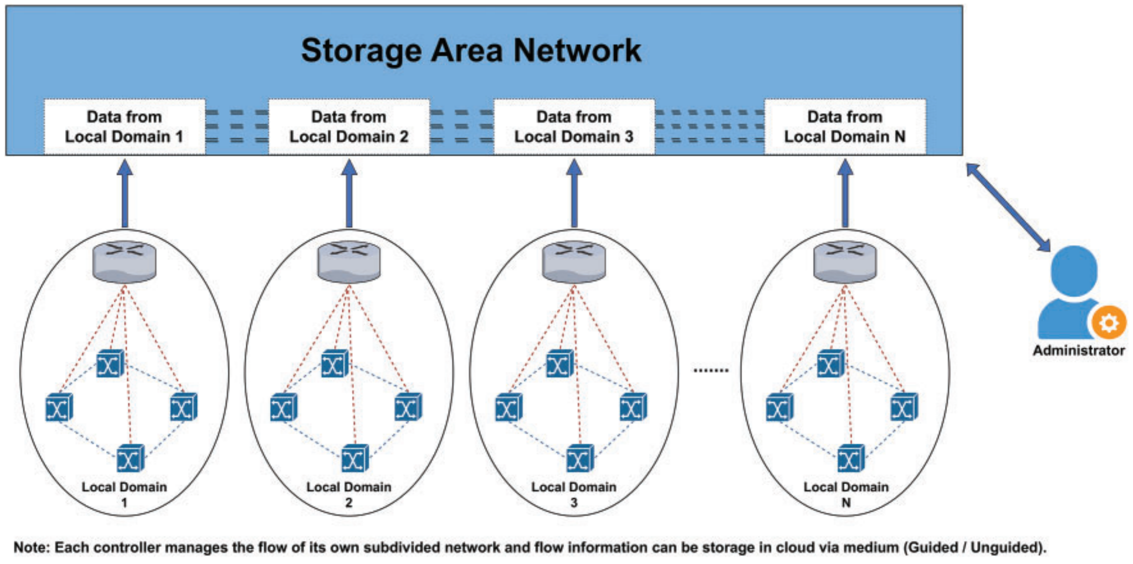


Figure 7: Illustration of the suggested technique's initial procedural step in a flow diagram

4.2 Procedural Step#2

The SDN controller determines each server's HTTP request load (RPS) to prevent network server overload carried by HTTP requests. The Grafana application is launched on the HTTP server and configured to modulate the data source in Prometheus format; to accomplish this, Grafana gives the required server logs access to the SDN controller when combined with Prometheus as the data source. Each server's RPS and active connection details are automatically retrieved every 10 milliseconds by designing a server load module at the POX controller using core and Psutil as Python library functions to fetch data by eliminating the need for lengthy calculation and Python coding. The following is the summary algorithm shown in Table 2.

Table 2: Procedural steps for evaluation of RPS matrices

Sr. no.	Procedural step explanation

1	def Integrate_ HTTP servers_RPS matrices_with_POXContoller ():
	    PSUTIL_setup ()
	    PSUTIL_Integrate_with_POXController ()
	    Endpoint_Creation_in_POXController_for_handling_RPS matrices ()
	    APACHE_Setup ()
	    GRAFANA_Run ()
	    Data_source_addition_of_PROMETHEUS ()
	    Construct_PromQL_query_manager_for_RPS_matrices ()

2	def Integrate_ HTTP servers_RPS matrices_with_POXContoller (): Print ("Amending the POX controller module to retrieve each HTTP server's RPS matrices data")
3	def PSUTIL_setup ():

(Continued)

Table 2 (continued)

Sr. no.	Procedural step explanation
	Print ("The Python library function that is installed and integrated with the POX controller is called PSUTIL")
4	def PSUTIL_Integrate_with_POXController (): Print ("With the POX controller installed, PSUTIL is the Python library function that is integrated")
5	def Endpoint_Creation_in_POXController_for_handling_RPS matrices (): Print ("To expose all of each server's RPS matrix values to the POX controller, an endpoint is constructed using PSUTIL")
6	def APACHE_Setup (): Print ("Every HTTP server has the APACHE program installed")
7	def GRAFANA_Run (): Print ("On every server, the GRAFANA application is started")
8	def Data_source_addition_of_PROMETHEUS (): Print ("To select the data source for PROMETHEUS to extract the HTTP server's data logs, the GRAFANA application is run on each server and its browser is extracted")
9	def Construct_PromQL_query_manager_for_RPS_matrices (): Print ("To query the RPS matrix (i.e., needed parameters, default time, memory detail, etc.) of each server, Prom_QL queries are built")

4.3 Procedural Step#3

After forming the SDN network on the Mininet tool, the controller calculates the HTTP request load (RPS) for each server in the network. If the RPS value for any request severing sever is greater than the reference load threshold value ($S_{\_Load}$), the following conditions are met in the proposed algorithm:

```
if (Request load of requested server > = Server load)
    {
    Remove the requested server from the available servers list;
    For five times:
        If (Server search with lesser request load value = Successful)
        {
            Move the load of HTTP requests to that server;
            break the loop;
        }
        else if (Server search with the fewest connections active = Successful)
        {
            Move the load of HTTP requests to that server;
            break the loop;
        }
```

```
            else if (Server search with slowest response time = Successful)
            {
                Move the load of HTTP requests to that server;
                break the loop;
            }
            else
            {
                Print "The optimal server was not found";
             }
        }
    else
        {
          The requested server is given a new HTTP flow;
        }
```

The procedure of Step 3 is illustrated in Fig. 8.

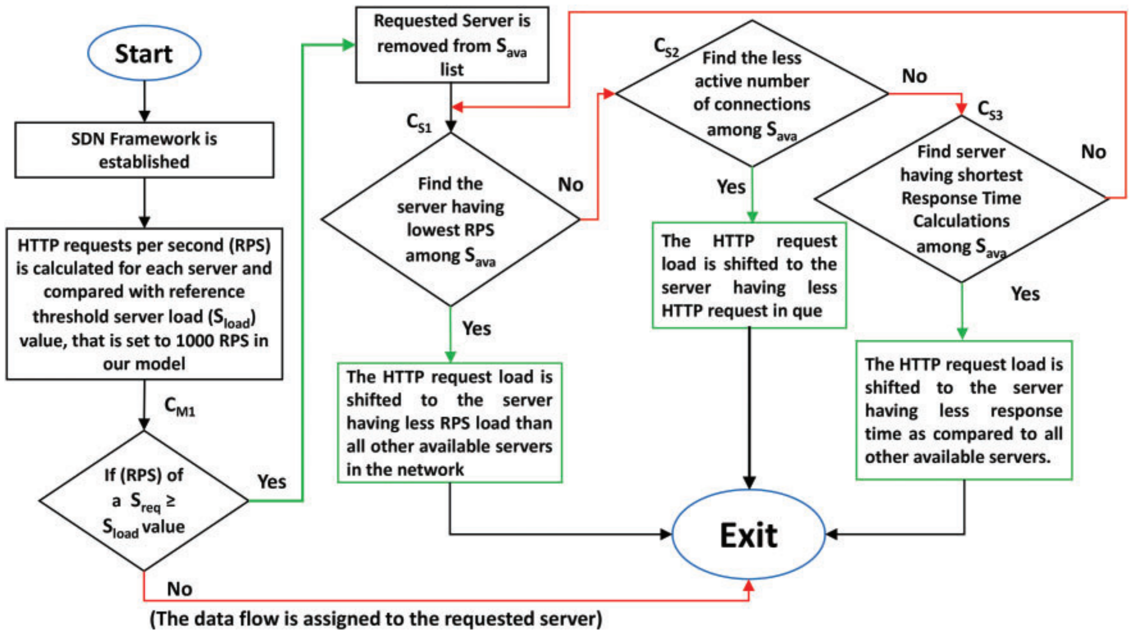

Start
SDN Framework is established
HTTP requests per second (RPS) is calculated for each server and compared with reference threshold server load (S_{load}) value, that is set to 1000 RPS in our model
C_{M1}
If (RPS) of a S_{req} ≥ S_{load} value
Yes
No
(The data flow is assigned to the requested server)
Requested Server is removed from S_{ava} list
C_{S1}
Find the server having lowest RPS among S_{ava}
No
Yes
The HTTP request load is shifted to the server having less RPS load than all other available servers in the network
C_{S2}
Find the less active number of connections among S_{ava}
No
Yes
The HTTP request load is shifted to the server having less HTTP request in que
C_{S3}
Find server having shortest Response Time Calculations among S_{ava}
No
Yes
The HTTP request load is shifted to the server having less response time as compared to all other available servers.
Exit

Figure 8: The second and third procedural steps of the suggested technique are represented in a flow diagram

4.4 A Comparison between Traditional Server Load Balancing Algorithms and the Proposed (ISSLM) Algorithm for Efficient Server Load Balancing

This portion includes an algorithm, as shown in Table 3, to perform server load balancing with the research technique (ISSLM) algorithm.

Table 3: Algorithm to perform server load balancing with ISSLM

Steps	Explanation
Initialization:	
	"total_Unavailable_Servers" ← Count of currently unavailable servers in the network. "server_With_MinLoad" ← Server with minimum HTTP request load in the network. The "chosen_Optimal_Server" ← server was selected via the ISSLM algorithm for new HTTP request management. "incoming_HTTP_Request_Batch" ← The SDN controller will manage A new batch of HTTP requests.
Algorithm Workflow:	
1	Evaluate if "incoming_HTTP_Request_Batch" equals "total_Unavailable_Servers." If true, proceed to Step 2. If false, move to Step 4.
2	Identify the optimal server with the lowest load among available servers: "available_Servers" = [active_Server_List] def select Server_With_MinLoad(available_Servers) return min(available_Servers, key=lambda server: available_Servers.get(server)) **Assign** "chosen_Optimal_Server" = select Server_With_MinLoad(available_Servers)
3	If the condition in Step 2 is not satisfied, select the optimal server with the fewest active connections: "connection_Servers" = [active_Server_List] def select Least_Connections_Server(connection_Servers): return min(connection_Servers, key=lambda server: connection_Servers.get(server)) **Assign** "chosen_Optimal_Server" = select LeastConnectionsServer (connectionServers)

4	If neither of the previous conditions are satisfied, select the optimal server with the quickest response time:
	"avail_Servers" = [active_Server_List]
	def Server_RespTime(avail_Servers):
	  startTime = get_current_time()
	  response = send_request_to_server(availServers)
	  endTime = get_current_time()
	  responseTime = endTime – startTime
	  print(f"Server response time: {responseTime} milliseconds")
	def Fastest Resp_Server(responseTime):
	  return min(responseTime, key=lambda server: responseTime.get(server))
	Assign "chosen_Optimal_Server" = FastestRespServer(ServerRespTime (availServers))

4.4.1 Algorithm for Segregating the Control Plane of Large SDN-Based DCN

This portion includes an algorithm, as shown in Table 4, to segregate the control plane of large SDN-based DCN with the research technique (ISSLM) algorithm.

Table 4: Algorithm to perform server load balancing with ISSLM

```
Sr. no.   Explanation
1         class NetworkManager:
2             def __init__(self, identifier, http_RequestCount):
                  self.identifier = identifier
                  self.http_RequestCount = http_RequestCount
                  self.qualityMetrics = {}
3             def monitor_subnetwork(self):
                  self.qualityMetrics = {"Delay-EndToEnd": "ms," "Throughput": "Gbps,"
                  "DataTransfer": "GB"}
4             def balance_http_load_with_ISSLM(self):
                  print(f"Applying ISSLM to balance load for controller {self.identifier} handling
                  {self.httpRequestCount} HTTP requests.")
5             def store_qos_metrics(self, storage_Area_Network):
                  storage_Area_Network[self.identifier] = self.qualityMetrics
6         class SAN:
7             def distribute_sdn_control(http_Request_Distributions):
                  storage_Network = {}
                  for index, http_RequestCount in enumerate(http_Request_Distributions):
                      manager = Network_Manager(index, http_RequestCount)
                      manager.balance_http_load_with_ISSLM()
                      manager.monitor_subnetwork()
                      manager.store_qos_metrics(storage_Network)
                  return storage_Network
```

5 Results and Discussion

For the evaluation of the quality metrics parameters (maximum available bandwidth: $B_{d(max)}$, Throughput: T_h, Data transfer: $D_{Transfer}$, and end-to-end delay: D_{E-E}, etc.), the SDN-based network is simulated in three different configurations: (1) A single POX controller-based SDN controller arrangement designed on Mininet without any server load management algorithm for managing network data traffic. (2) A single POX controller-based SDN controller arrangement designed on Mininet with a proposed server load management algorithm (ISSLM) for managing network traffic. (3) Deployment of SDN in logically distributed controlled framework arrangement designed on Mininet with proposed server load management algorithm (ISSLM) for managing network data traffic. Fig. 9 represents the virtual network configurations designed on the Mininet tool for the first two cases of simulations.

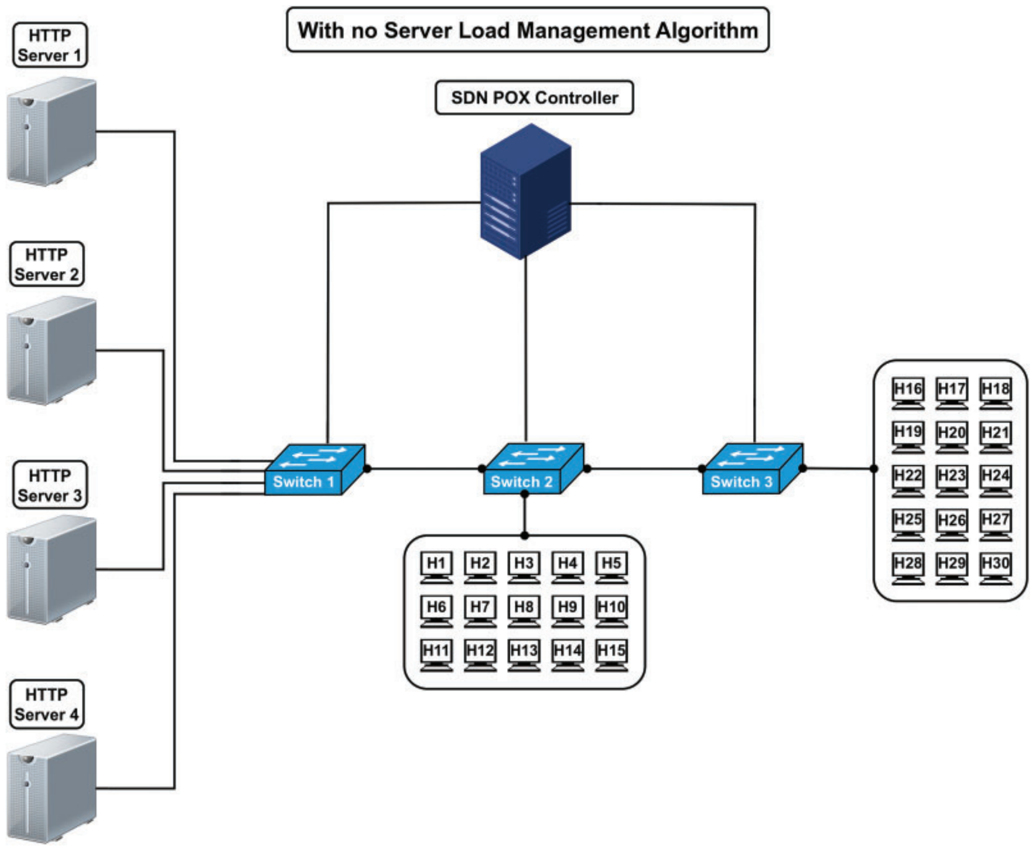


Figure 9: Network topology selected for obtaining QoS parameter results in the first portion of the simulation

5.1 Case-A: Evaluation of the Quality Metrics Parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, $D_{E\text{-}E}$, etc.) in a Single POX Controller-Based SDN Controller Arrangement Designed on Mininet without Any Server Load Management Algorithm for Managing Network Data Traffic

The user-defined network designed on Mininet, as shown in Fig. 9, comprises thirty HTTP request-generating hosts ($H_{st\text{-}1}$ to $H_{st\text{-}30}$), three OpenFlow switches (S_{w1}, S_{w2}, S_{w3}), four servers ($S_{er\text{-}1}$, $S_{er\text{-}2}$, $S_{er\text{-}3}$, and $S_{er\text{-}4}$), and single POX controller (C_{Ps}) that is responsible for managing the HTTP request load that is directed towards the servers. In this virtualized network environment, the servers are accordingly assigned the IP addresses 100.0.0.1, 100.0.0.2, 100.0.0.3, and 100.0.0.4. The QoS metrics ($B_{d(max)}$, T_h, $D_{Transfer}$, $D_{E\text{-}E}$, etc.) obtained under normal working conditions (200 randomly generated HTTP requests from hosts $H_{st\text{-}1}$–$H_{st\text{-}30}$ in our test case) include a maximum bandwidth ($B_{d(max)}$) of 6.5 Gb/s, a throughput (T_h) of 5.6152 Gb/s, and a data transfer ($D_{Transfer}$) volume of 7.019 Gbytes. However, during the simulation time of 10 s, a high volume of HTTP requests of 20,000 are made from randomly available Hosts ($H_{st\text{-}1}$ to $H_{st\text{-}30}$), and every request is exclusively sent to $S_{er\text{-}2}$ only. Because in this scenario there is no server load management technique applied on the Controller (C_{Ps}), it makes the $S_{er\text{-}2}$ get overloaded and reach a bottleneck state. As a result, QoS parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, $D_{E\text{-}E}$, etc.) deteriorate. Fig. 10 shows the QoS parameters for maximum bandwidth ($B_{d(max)}$) and data transfer ($D_{Transfer}$) related to the link between $S_{er\text{-}2}$ and the initiating hosts using the Iperf tool. Equations (Eq. (1)–(4)) are used to obtain the values of throughput (T_h), the percentage drop in data transfer (%$D_{Transfer}$), the percentage increase in server load ($\%I_{SL}$), and end-to-end Delay ($D_{E\text{-}E}$). Where the Round-Trip Delay (RTD) is obtained from the Iperf utility by using its h_{ping} directory.

$$T_h = \frac{Data\_Transfer\ in\ (Gbytes)}{Time(s)} \tag{1}$$

$$\%I_{SL} = 100 - \left(\frac{100 * (Bd(max)\ under\ loaded\ conditions)}{Bd\ (max)\ under\ normal\ conditions}\right) \tag{2}$$

$$\%D_{Transfer} = 100 - \left(\frac{100 * (Data\_Transfer\ under\ loaded\ conditions)}{Data\_Transfer\ under\ normal\ conditions}\right) \tag{3}$$

$$D_{E-E} = \left(\frac{RTD}{2}\right) \tag{4}$$

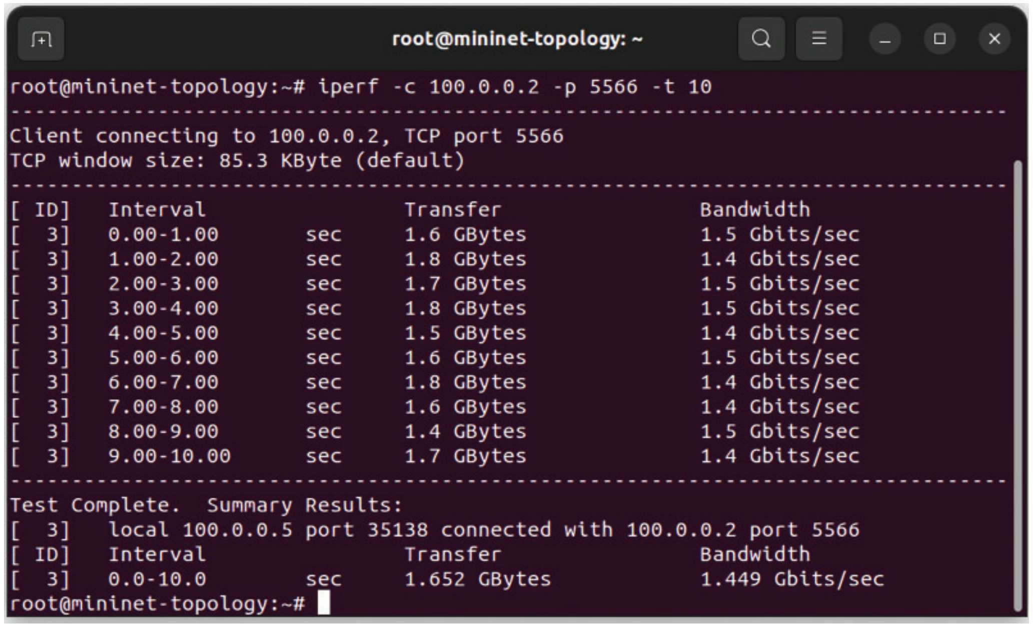

```
root@mininet-topology:~# iperf -c 100.0.0.2 -p 5566 -t 10
------------------------------------------------------------------------------
Client connecting to 100.0.0.2, TCP port 5566
TCP window size: 85.3 KByte (default)
------------------------------------------------------------------------------
[ ID]   Interval            sec     Transfer                Bandwidth
[  3]   0.00-1.00           sec     1.6 GBytes              1.5 Gbits/sec
[  3]   1.00-2.00           sec     1.8 GBytes              1.4 Gbits/sec
[  3]   2.00-3.00           sec     1.7 GBytes              1.5 Gbits/sec
[  3]   3.00-4.00           sec     1.8 GBytes              1.5 Gbits/sec
[  3]   4.00-5.00           sec     1.5 GBytes              1.4 Gbits/sec
[  3]   5.00-6.00           sec     1.6 GBytes              1.5 Gbits/sec
[  3]   6.00-7.00           sec     1.8 GBytes              1.4 Gbits/sec
[  3]   7.00-8.00           sec     1.6 GBytes              1.4 Gbits/sec
[  3]   8.00-9.00           sec     1.4 GBytes              1.5 Gbits/sec
[  3]   9.00-10.00          sec     1.7 GBytes              1.4 Gbits/sec
------------------------------------------------------------------------------
Test Complete.  Summary Results:
[  3]   local 100.0.0.5 port 35138 connected with 100.0.0.2 port 5566
[ ID]   Interval                    Transfer                Bandwidth
[  3]   0.0-10.0            sec     1.652 GBytes            1.449 Gbits/sec
root@mininet-topology:~#
```

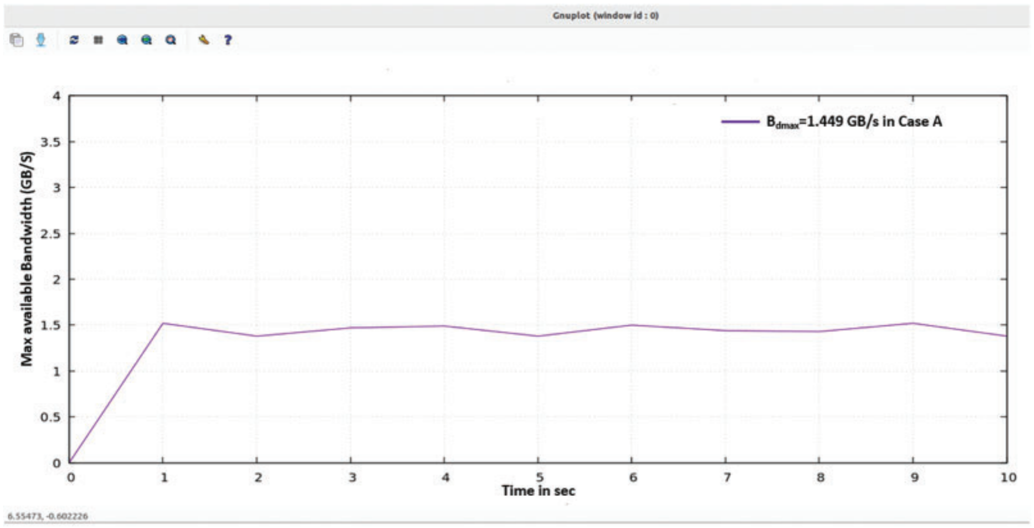
Figure 10: Results of quality parameters obtained through Iperf utility using a single controller (C_{Ps}) without load management

Table 5 represents the QoS parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, $D_{E\text{-}E}$, etc.) for Case A with integration of Iperf utility with the Mininet environment over 10 s. When all 20,000 requests are sent to a $S_{er\text{-}2}$ without the use of load balancing, there is a noticeable increase in the server load ($\%I_{SL}$) as compared to the normal conditions. The percentage of data transfer ($\%D_{Transfer}$) is also drastically decreased compared to normal conditions (i.e., 200 randomly generated HTTP requests from hosts $H_{st\text{-}1}$–$H_{st\text{-}30}$).

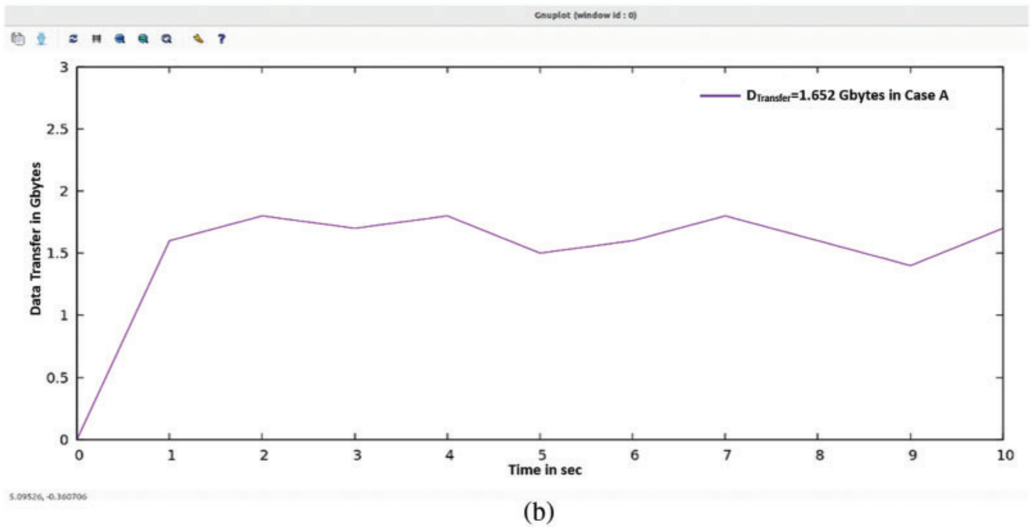
Table 5: QoS parameters of Ser-2 using a single controller (C_{Ps}) without a Load Balancing Algorithm

Time in (S_{econds})	$S_{requested}$	$B_{d(max)}$ in (Gb/s)	$D_{Transfer}$ in (G-bytes)	T_h in (Gb/s)	$\%I_{SL}$	$\%D_{Transfer}$
10	S_2	1.449	1.652	1.322	77.70%	76.49%

Line graphs of QoS parameters ($B_{d(max)}$ and $D_{Transfer}$) for S_{er-2} using the Gnu Plot tool are represented in Fig. 11a,b.



(a)



(b)

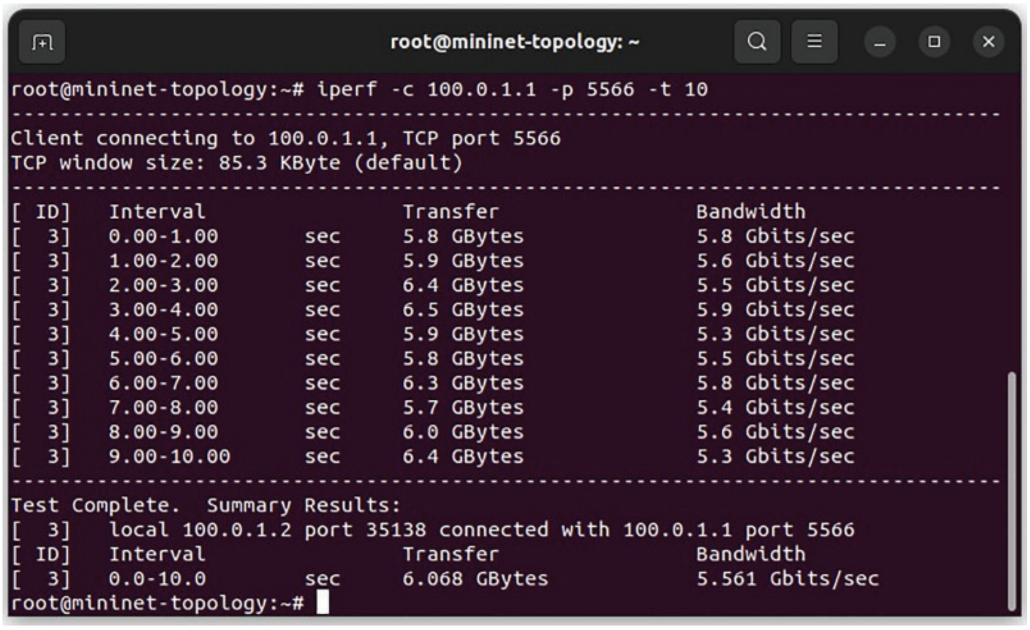
Figure 11: Line graphs of ($Bd_{(max)}$ and $D_{Transfer}$) for Server_2 without load management. (a) $B_{d(max)}$ using a single controller (C_{Ps}) without load management. (b) $D_{Transfer}$ using a single controller (C_{Ps}) without load management

5.1.1 Summarizing QoS Parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, $D_{E\text{-}E}$, etc.) Result of Case A

When a network topology shown in Fig. 9 is simulated to obtain the QoS parameters ($B_{d(max)}$, Th, $D_{Transfer}$, $D_{E\text{-}E}$, etc.) results with only a single controller without a load management algorithm, the overall network QoS is drastically decreased. The B_d decreased to the value (from 6.5 to 1.449 Gb/s). The T_h value is declined from (5.612 to 1.322 Gb/s). The $D_{Transfer}$ is also reduced from (7.019 to 1.652 Gbytes), and a Delay of (262.5 ms) is induced in the selected network. The percentage load on the network servers (%I_{SL}) is increased to (77.70%). The overall percentage reduction in data transfer (%$D_{Transfer}$) is (76.49%).

5.2 Case-B: Evaluation of the Quality Metrics Parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, $D_{E\text{-}E}$, etc.) in a Single POX Controller-Based SDN Controller Arrangement Designed on Mininet with Proposed Server Load Management Algorithm (ISSLM) for Managing Network Data Traffic

In this section, the proposed algorithm (ISSLM) in the form of Python script is loaded on the POX controller, which performs server load management under the direction of (the ISSLM) algorithm. In this mode, all the HTTP requests generated from randomly available hosts ($H_{st\text{-}1}$ to $H_{st\text{-}30}$) are first countered by an SDN controller that performs load balancing on network servers (S_{er1}, S_{er2}, $S_{er3,}$ and S_{er4}) with the aid of the ISSLM algorithm. In ten second simulation period, 20,000 HTTP requests are directed only toward the POX controller. Using the Iperf utility, the QoS parameters for Case B are obtained and shown in Fig. 12.



```
root@mininet-topology:~# iperf -c 100.0.1.1 -p 5566 -t 10
------------------------------------------------------------------------------------
Client connecting to 100.0.1.1, TCP port 5566
TCP window size: 85.3 KByte (default)
------------------------------------------------------------------------------------
[ ID]   Interval          Transfer          Bandwidth
[  3]   0.00-1.00    sec  5.8 GBytes        5.8 Gbits/sec
[  3]   1.00-2.00    sec  5.9 GBytes        5.6 Gbits/sec
[  3]   2.00-3.00    sec  6.4 GBytes        5.5 Gbits/sec
[  3]   3.00-4.00    sec  6.5 GBytes        5.9 Gbits/sec
[  3]   4.00-5.00    sec  5.9 GBytes        5.3 Gbits/sec
[  3]   5.00-6.00    sec  5.8 GBytes        5.5 Gbits/sec
[  3]   6.00-7.00    sec  6.3 GBytes        5.8 Gbits/sec
[  3]   7.00-8.00    sec  5.7 GBytes        5.4 Gbits/sec
[  3]   8.00-9.00    sec  6.0 GBytes        5.6 Gbits/sec
[  3]   9.00-10.00   sec  6.4 GBytes        5.3 Gbits/sec
------------------------------------------------------------------------------------
Test Complete.  Summary Results:
[  3]   local 100.0.1.2 port 35138 connected with 100.0.1.1 port 5566
[ ID]   Interval          Transfer          Bandwidth
[  3]   0.0-10.0     sec  6.068 GBytes      5.561 Gbits/sec
root@mininet-topology:~#
```

Figure 12: Results of quality parameters obtained through Iperf utility using a single controller (C_{Ps}) with ISSLM algorithm

Table 6 represents the QoS parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, $D_{E\text{-}E}$, etc.) for Case B with integration of Iperf utility with Mininet environment over 10 s. As it is evident from Table 6 and Fig. 12, the network quality parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, $D_{E\text{-}E}$) are greatly increased as compared to Case A when all 20,000 HTTP requests are controlled by the controller equipped with a proposed algorithm (ISSLM). The ($B_{d(max)}$) increases to 5.561 from 1.449 Gb/s. The T_h improves to 4.854 from 1.322 GB/s. There is also significant increase in $D_{Transfer}$, going from 1.652 G-bytes to 6.068 G-bytes. Regarding Eqs. (2) and (3), the (%$D_{Transfer}$) and (%I_{SL}) drop from 76.49% to 13.55% and 77.70% to 14.44%, respectively.

Line graphs of QoS parameters ($B_{d(max)}$ and $D_{Transfer}$) for Case B using the Gnu Plot tool are represented in Figs. 13 and 14.

Table 6: QoS parameters using a single controller (C_{Ps}) with a Load Balancing Algorithm (ISSLM)

Node	Time	$B_{d(max)}$	T_h	$D_{Transfer}$	$\%I_{SL}$	$\%D_{Transfer}$	$D_{(E-E)}$
QoS parameter extraction with Cs under ISSLM	10 s	5.561 Gb/s	4.854 Gb/s	6.068 G-bytes	14.44%	13.55%	36.7 ms
QoS parameter extraction with Cs without ISSLM	10 s	1.449 Gb/s	1.322 Gb/s	1.652 G-bytes	77.70%	76.49%	513.5 ms

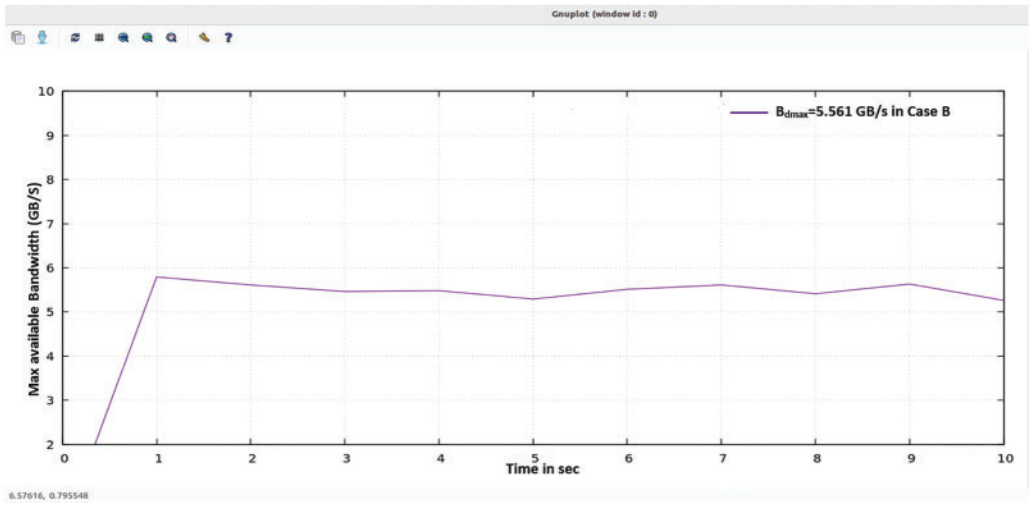


Figure 13: $B_{d(max)}$ using a single controller (C_{Ps}) with load management algorithm (ISSLM)

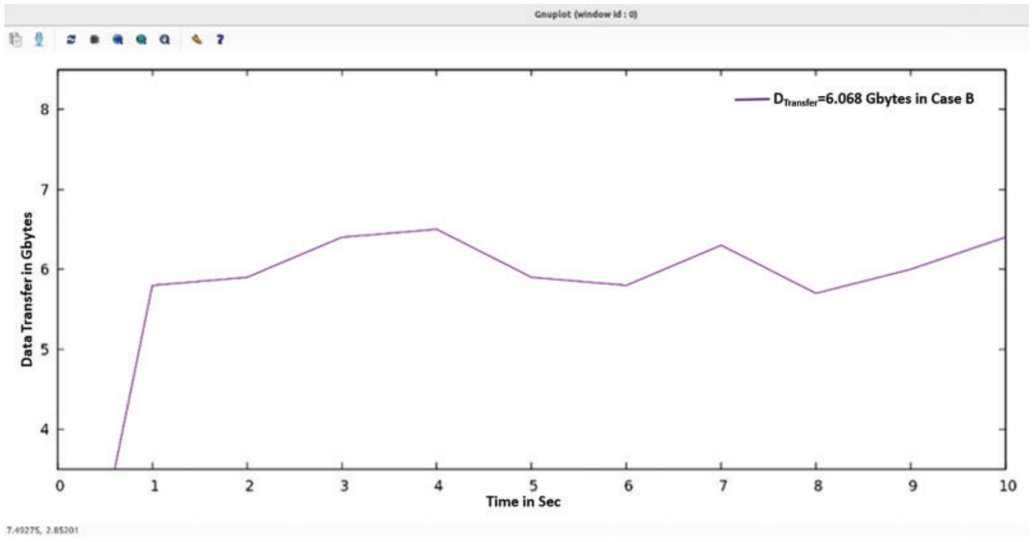


Figure 14: $D_{Transfer}$ using a single controller (C_{Ps}) with load management algorithm (ISSLM)

5.2.1 *Summarizing QoS Parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, $D_{E\text{-}E}$, etc.) Result of Case B*

When a network topology shown in Fig. 9 is simulated to obtain the QoS parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, $D_{E\text{-}E}$, etc.) results with only a single controller with the proposed load management algorithm (ISSLM), the overall QoS of the network is increased as compared to Case A. The B_d is maximized to the value (from 1.449 to 5.561 Gb/s). The T_h value is increased from (1.322 to 4.854 Gb/s). The $D_{Transfer}$ is also incremented from (1.652 to 6.068 Gbytes), and Delay is reduced to (513.5 to 36.7 ms) in the selected network. The percentage load on the network servers (%I_{SL}) is decreased up to (from 77.70% to 14.44%). The overall percentage reduction in data transfer (%$D_{Transfer}$) is also less in Case B (76.49% to 13.55%).

5.3 *Case-C: Evaluation of the Quality Metrics Parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, $D_{E\text{-}E}$, etc.) with Deployment of SDN in Logically Distributed Controlled Framework Arrangement Designed on Mininet with Proposed Server Load Management Algorithm (ISSLM) for Managing Network Data Traffic*

In this case, the SDN architecture is framed in a logically distributed controlled arrangement, as shown in Fig. 15. Thirty hosts ($H_{st\text{-}1}$ to $H_{st\text{-}30}$) are divided into four logically distributed SDN-based sub-networks (L_{d1}, L_{d2}, L_{d3}, and L_{d4}), each of which is overseen by a separate dedicated controller (C_{d1}, C_{d2}, C_{d3}, and C_{d4}). Now, instead of handling 20,000 HTTP requests by the single controller (C_{Ps}), the C_{d1} and C_{d3}, handle 6000 HTTP requests apiece that come from network devices in sub-networks L_{d1} and L_{d3}, respectively, using a proposed algorithm (ISSLM). Similarly, 4000 HTTP requests apiece are managed by C_{d2} and C_{d4} and directed from network devices inside L_{d2} and L_{d4} using the ISSLM algorithm with the help of logic presented in Table 4. Higher-level controllers C_{p1} and C_{p2} are responsible for overall network flow control management and fault tolerance.

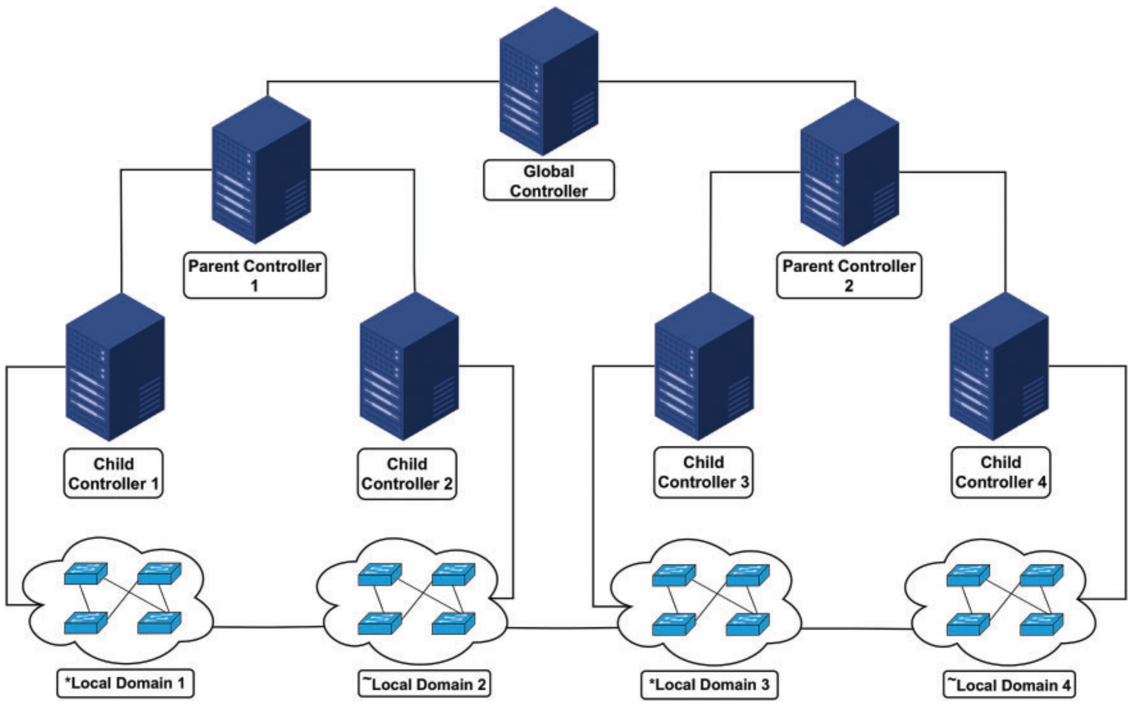
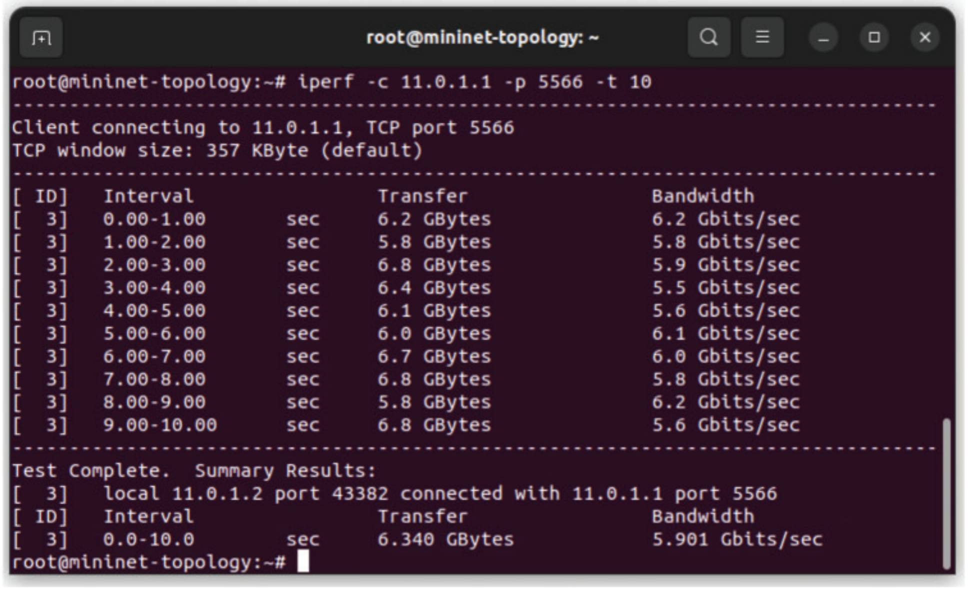


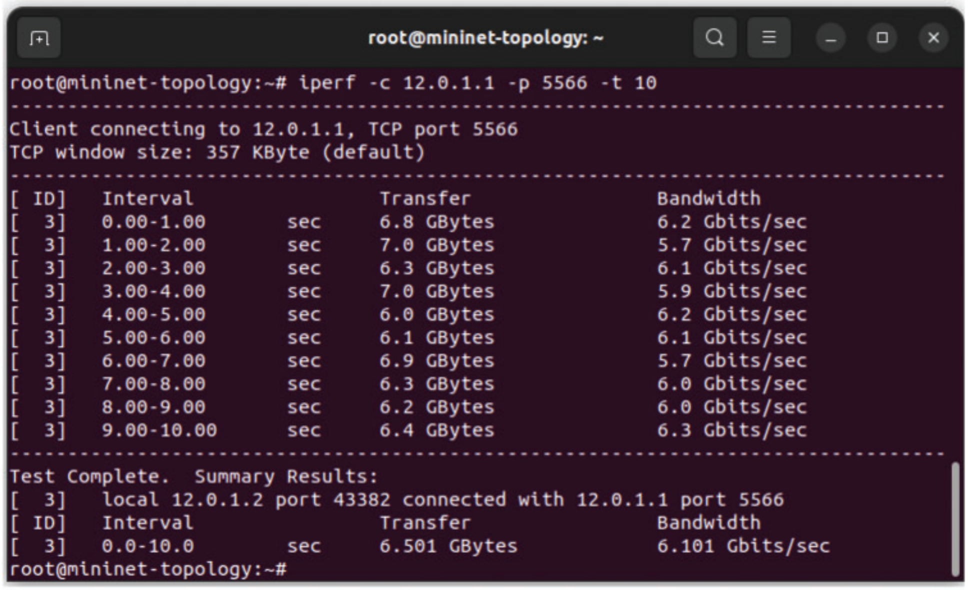
Figure 15: The proposed network configuration for deploying SDN in large DCNs

The subnetwork controllers (C_{d1}, C_{d2}, C_{d3} and C_{d4}) have an IP address as (11.0.1.1, 12.0.1.1, 13.0.1.1, and 14.0.1.1), respectively. For subnetworks (L_{d1}, L_{d2}, L_{d3} and L_{d4}), the corresponding IP addresses are (11.0.1.2, 12.0.1.2, 13.0.1.2 and 14.0.1.2), respectively. Fig. 16 illustrates the values of ($B_{d(max)}$, T_h, and $D_{Transfer}$) in all sub-divided networks (L_{d1}, L_{d2}, $L_{d3,}$ and L_{d4}) using the Iperf utility.



```
root@mininet-topology:~# iperf -c 11.0.1.1 -p 5566 -t 10
------------------------------------------------------------------------------------
Client connecting to 11.0.1.1, TCP port 5566
TCP window size: 357 KByte (default)
------------------------------------------------------------------------------------
[ ID]   Interval            Transfer               Bandwidth
[  3]   0.00-1.00      sec     6.2 GBytes             6.2 Gbits/sec
[  3]   1.00-2.00      sec     5.8 GBytes             5.8 Gbits/sec
[  3]   2.00-3.00      sec     6.8 GBytes             5.9 Gbits/sec
[  3]   3.00-4.00      sec     6.4 GBytes             5.5 Gbits/sec
[  3]   4.00-5.00      sec     6.1 GBytes             5.6 Gbits/sec
[  3]   5.00-6.00      sec     6.0 GBytes             6.1 Gbits/sec
[  3]   6.00-7.00      sec     6.7 GBytes             6.0 Gbits/sec
[  3]   7.00-8.00      sec     6.8 GBytes             5.8 Gbits/sec
[  3]   8.00-9.00      sec     5.8 GBytes             6.2 Gbits/sec
[  3]   9.00-10.00     sec     6.8 GBytes             5.6 Gbits/sec
------------------------------------------------------------------------------------
Test Complete.  Summary Results:
[  3]   local 11.0.1.2 port 43382 connected with 11.0.1.1 port 5566
[ ID]   Interval            Transfer               Bandwidth
[  3]   0.0-10.0       sec     6.340 GBytes           5.901 Gbits/sec
root@mininet-topology:~# 
```

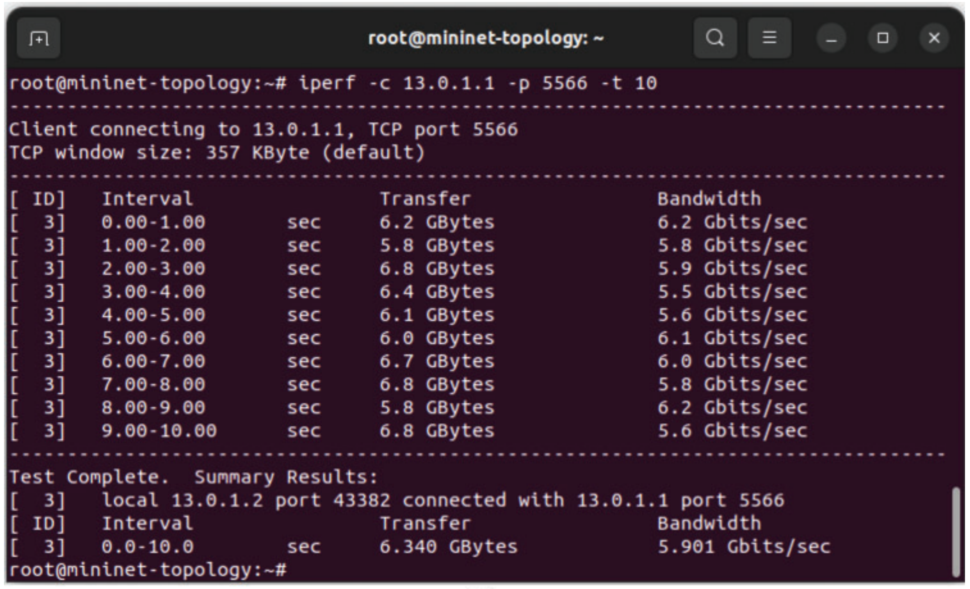
(a)



```
root@mininet-topology:~# iperf -c 12.0.1.1 -p 5566 -t 10
------------------------------------------------------------------------------------
Client connecting to 12.0.1.1, TCP port 5566
TCP window size: 357 KByte (default)
------------------------------------------------------------------------------------
[ ID]   Interval            Transfer               Bandwidth
[  3]   0.00-1.00      sec     6.8 GBytes             6.2 Gbits/sec
[  3]   1.00-2.00      sec     7.0 GBytes             5.7 Gbits/sec
[  3]   2.00-3.00      sec     6.3 GBytes             6.1 Gbits/sec
[  3]   3.00-4.00      sec     7.0 GBytes             5.9 Gbits/sec
[  3]   4.00-5.00      sec     6.0 GBytes             6.2 Gbits/sec
[  3]   5.00-6.00      sec     6.1 GBytes             6.1 Gbits/sec
[  3]   6.00-7.00      sec     6.9 GBytes             5.7 Gbits/sec
[  3]   7.00-8.00      sec     6.3 GBytes             6.0 Gbits/sec
[  3]   8.00-9.00      sec     6.2 GBytes             6.0 Gbits/sec
[  3]   9.00-10.00     sec     6.4 GBytes             6.3 Gbits/sec
------------------------------------------------------------------------------------
Test Complete.  Summary Results:
[  3]   local 12.0.1.2 port 43382 connected with 12.0.1.1 port 5566
[ ID]   Interval            Transfer               Bandwidth
[  3]   0.0-10.0       sec     6.501 GBytes           6.101 Gbits/sec
root@mininet-topology:~#
```

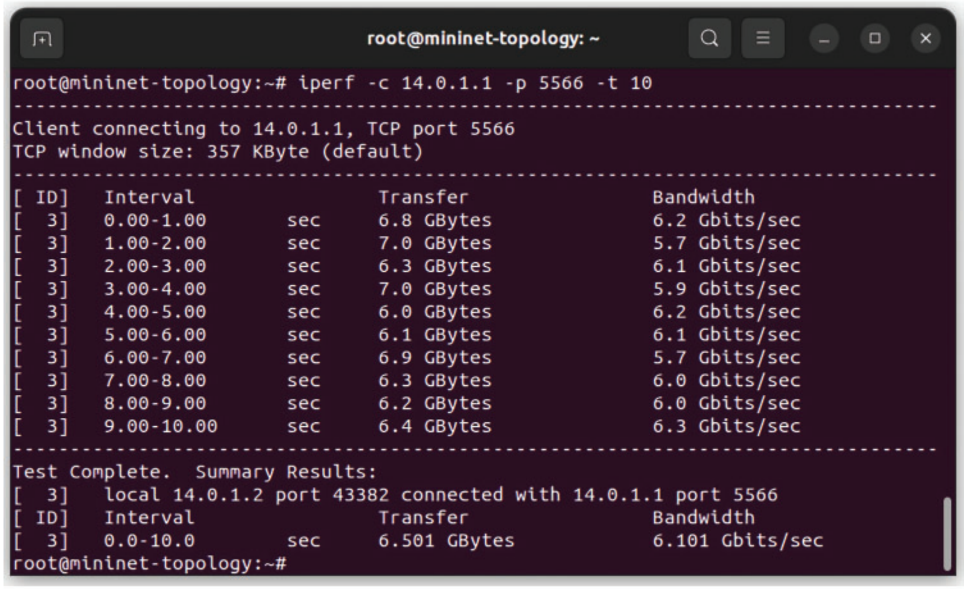
(b)

Figure 16: (Continued)



```
root@mininet-topology:~# iperf -c 13.0.1.1 -p 5566 -t 10
------------------------------------------------------------------
Client connecting to 13.0.1.1, TCP port 5566
TCP window size: 357 KByte (default)
------------------------------------------------------------------
[ ID]   Interval          Transfer        Bandwidth
[  3]   0.00-1.00    sec   6.2 GBytes      6.2 Gbits/sec
[  3]   1.00-2.00    sec   5.8 GBytes      5.8 Gbits/sec
[  3]   2.00-3.00    sec   6.8 GBytes      5.9 Gbits/sec
[  3]   3.00-4.00    sec   6.4 GBytes      5.5 Gbits/sec
[  3]   4.00-5.00    sec   6.1 GBytes      5.6 Gbits/sec
[  3]   5.00-6.00    sec   6.0 GBytes      6.1 Gbits/sec
[  3]   6.00-7.00    sec   6.7 GBytes      6.0 Gbits/sec
[  3]   7.00-8.00    sec   6.8 GBytes      5.8 Gbits/sec
[  3]   8.00-9.00    sec   5.8 GBytes      6.2 Gbits/sec
[  3]   9.00-10.00   sec   6.8 GBytes      5.6 Gbits/sec
------------------------------------------------------------------
Test Complete.  Summary Results:
[  3]   local 13.0.1.2 port 43382 connected with 13.0.1.1 port 5566
[ ID]   Interval          Transfer        Bandwidth
[  3]   0.0-10.0     sec   6.340 GBytes    5.901 Gbits/sec
root@mininet-topology:~#
```

(c)



```
root@mininet-topology:~# iperf -c 14.0.1.1 -p 5566 -t 10
------------------------------------------------------------------
Client connecting to 14.0.1.1, TCP port 5566
TCP window size: 357 KByte (default)
------------------------------------------------------------------
[ ID]   Interval          Transfer        Bandwidth
[  3]   0.00-1.00    sec   6.8 GBytes      6.2 Gbits/sec
[  3]   1.00-2.00    sec   7.0 GBytes      5.7 Gbits/sec
[  3]   2.00-3.00    sec   6.3 GBytes      6.1 Gbits/sec
[  3]   3.00-4.00    sec   7.0 GBytes      5.9 Gbits/sec
[  3]   4.00-5.00    sec   6.0 GBytes      6.2 Gbits/sec
[  3]   5.00-6.00    sec   6.1 GBytes      6.1 Gbits/sec
[  3]   6.00-7.00    sec   6.9 GBytes      5.7 Gbits/sec
[  3]   7.00-8.00    sec   6.3 GBytes      6.0 Gbits/sec
[  3]   8.00-9.00    sec   6.2 GBytes      6.0 Gbits/sec
[  3]   9.00-10.00   sec   6.4 GBytes      6.3 Gbits/sec
------------------------------------------------------------------
Test Complete.  Summary Results:
[  3]   local 14.0.1.2 port 43382 connected with 14.0.1.1 port 5566
[ ID]   Interval          Transfer        Bandwidth
[  3]   0.0-10.0     sec   6.501 GBytes    6.101 Gbits/sec
root@mininet-topology:~#
```

(d)

Figure 16: QoS parametric values of four subdivided networks (L_{d1}, L_{d2}, L_{d3}, and L_{d4}) using Iperf utility. (a): Displays the QoS parametric values for (L_{d1}) using Iperf utility. (b): Displays the QoS parametric values for (L_{d2}) using the Iperf utility. (c): Displays the QoS parametric values for (L_{d3}) using Iperf utility. (d): Displays the QoS parametric values for (L_{d4}) using the Iperf utility

As evidenced by Table 7 and Fig. 16, the QoS parametric values in the proposed framework (logically distributed controlled arrangement) by implementing the proposed algorithm (ISSLM) are far greater than the quality requests obtained from the Case (A and B).

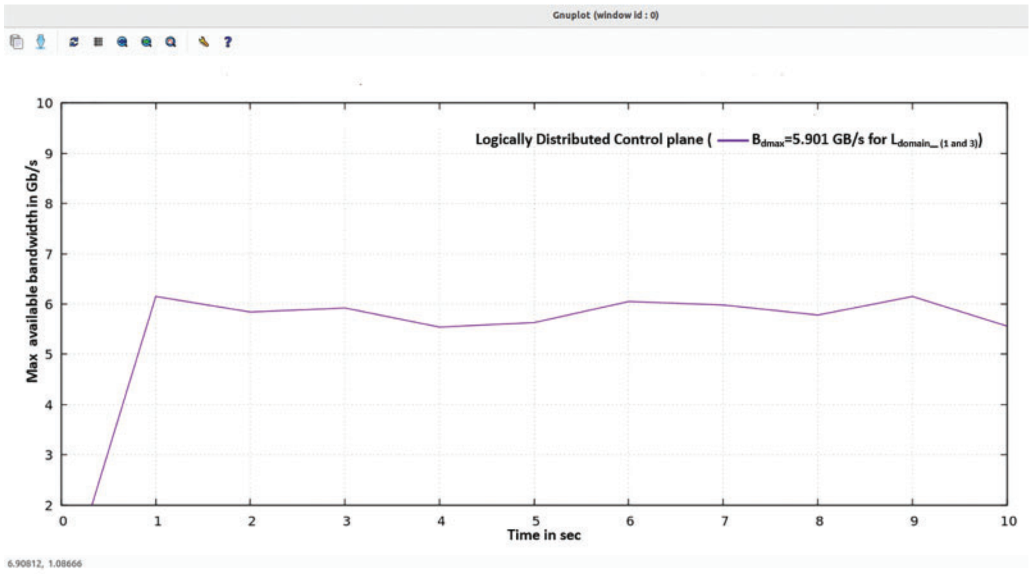
Table 7: Comparative analysis of QoS parametric values obtained in all three portions

Terminal	Time in (S_{econds})	$B_{d(max)}$ in (Gb/s)	T_h in (Gb/s)	$D_{Transfer}$ in (G-bytes)	$\%I_{SL}$	$\%D_{Transfer}$
QoS parameters of Case A	10	1.449	1.322	1.652	77.70%	76.49%
QoS parameters of Case B	10	5.561	4.854	6.068	14.44%	13.55%
QoS parameters for local domain (L_{d1})	10	5.901	5.072	6.340	9.21%	9.67%
QoS parameters for local domain (L_{d2})	10	6.101	5.201	6.501	6.14%	7.38%
QoS parameters for local domain (L_{d3})	10	5.901	5.072	6.340	9.21%	9.67%
QoS parameters for local domain (L_{d4})	10	6.101	5.201	6.501	6.14%	7.38%

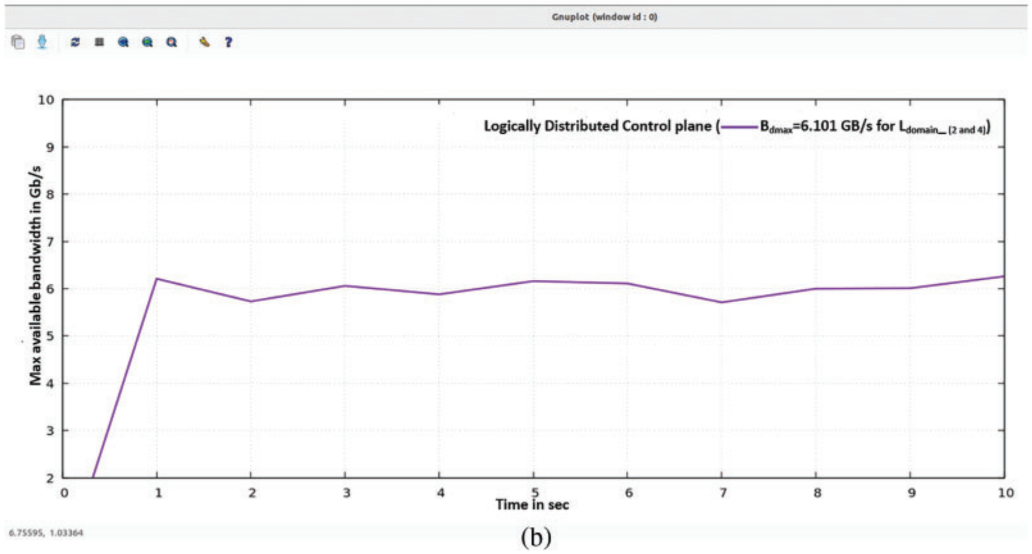
It is the dividend from Table 7 that ($B_{d(max)}$) for (L_{d1}, L_{d2}, $L_{d3,}$ and L_{d4}) has increased to (5.901, 6.101, 5.901, 6.101 Gb/s), respectively, from (5.561 Gb/s), as recorded under QoS parameters in Case B. Similarly (T_h) for each subnetwork has increased from 4.854 Gb/s (throughput as QoS parametric value in Case B) to (5.072, 5.201, 5.072, 5.201 Gb/s) for (L_{d1}, L_{d2}, L_{d3} and L_{d4}), respectively. ($D_{Transfer}$) for each subnetwork has increased from 6.068 Gbytes (Data transfer as QoS parametric value in Case B) to (6.340, 6.501, 6.340, 6.501 Gbytes) for (L_{d1}, L_{d2}, L_{d3} and L_{d4}), respectively. Moreover, the ($\%I_{SL}$) has dropped throughout the corresponding sub-networks from 14.44% (QoS parametric value in Case B) to (9.21%, 6.14%, 9.21%, 6.14%), for (L_{d1}, L_{d2}, $L_{d3,}$ and L_{d4}), respectively. The ($\%D_{Transfer}$) has dropped throughout the corresponding subnetworks from 13.55% (QoS parametric value in Case B) to (9.67%, 7.38%, 9.67%, 7.38%), for (L_{d1}, L_{d2}, $L_{d3,}$ and L_{d4}), respectively. These differences in $B_{d(max)}$and $D_{Transfer}$ across the segmented networks (L_{d1}, L_{d2}, L_{d3}, and L_{d4}) are graphically represented in line graph format using the Gnu plot in Fig. 17.

5.3.1 Summarizing QoS Parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, D_{E-E}, etc.) Result of Case C

When a network is framed in a logically distributed controlled environment, as shown in Fig. 15, the QoS parameters ($B_{d(max),}$ T_h, $D_{transfer}$, D_{E-E}, etc.) results are far more superior as compared to QoS parameters of Case (A & B). The B_d is maximized from (5.561 Gb/s) in Case B to (5.901, 6.101, 5.901, 6.101 Gb/s) for (L_{d1}, L_{d2}, L_{d3} and L_{d4}), respectively. The T_h value is increased from (4.854 Gb/s) in Case B to (5.072, 5.201, 5.072, 5.201 Gb/s) for (L_{d1}, L_{d2}, L_{d3} and L_{d4}), respectively. The $D_{transfer}$ is also incremented from (6.068 Gbytes) in Case B to (6.340, 6.501, 6.340, 6.501 Gbytes) for (L_{d1}, L_{d2}, L_{d3} and L_{d4}) respectively and Delay is reduced from (36.7 ms) in Case B to (3.57, 0.87, 3.57, 0.87 ms) for (L_{d1}, L_{d2}, L_{d3} and L_{d4}) respectively in the selected network. The percentage load on the network servers ($\%I_{SL}$) decreased from (14.44%) in Case B to (9.21%, 6.14%, 9.21%, 6.14%) for (L_{d1}, L_{d2}, L_{d3} and L_{d4}), respectively. The overall percentage reduction in data transfer ($\%D_{Transfer}$) is also less in Case C (9.67%, 7.38%, 9.67%, 7.38%) for (L_{d1}, L_{d2}, $L_{d3,}$ and L_{d4}), respectively.

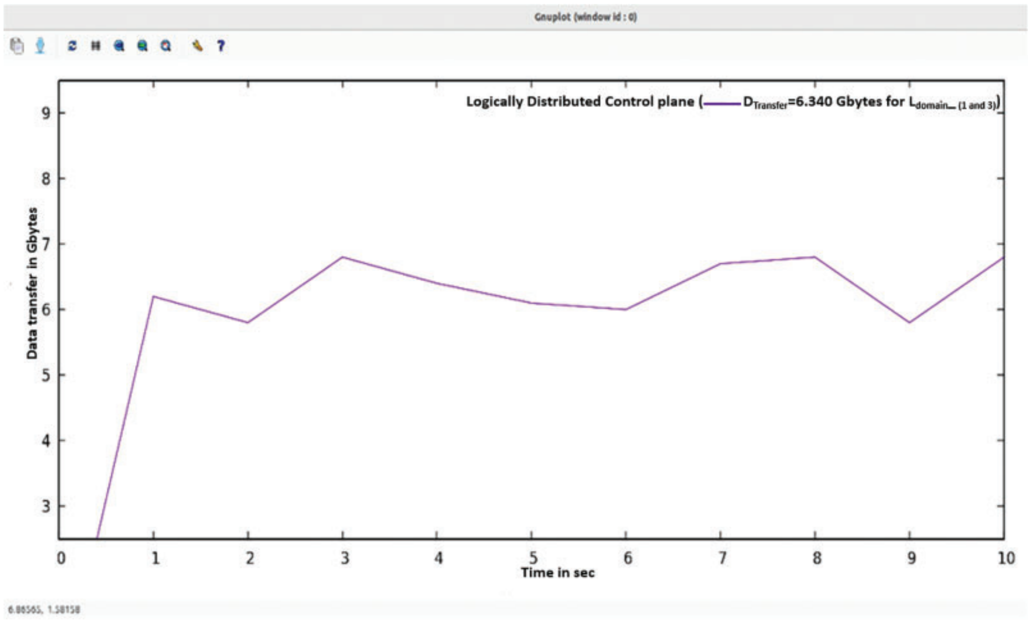


(a)

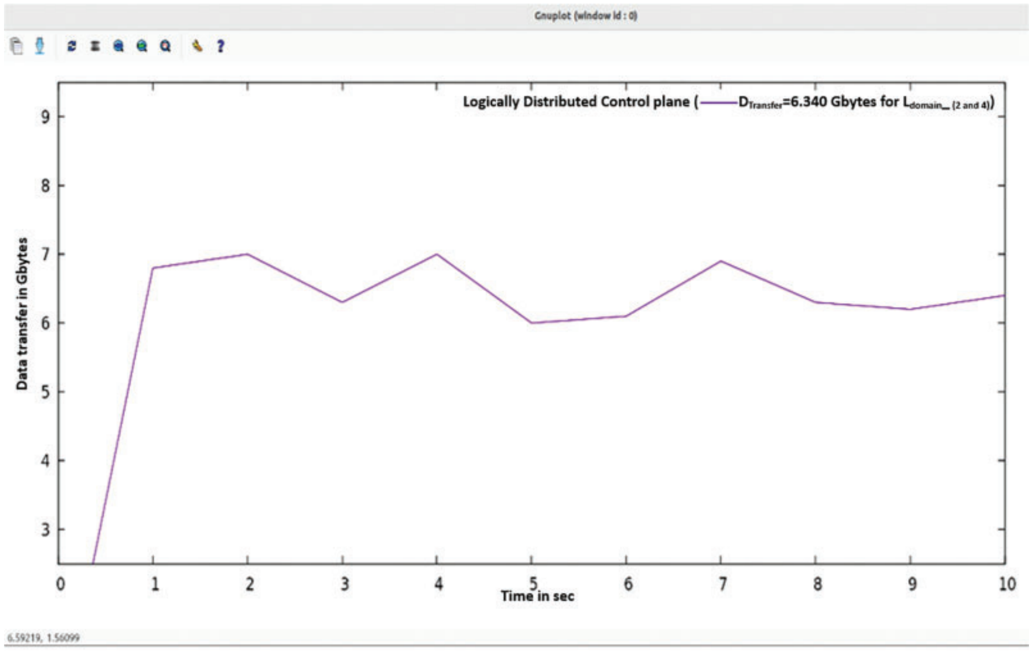


(b)

Figure 17: (Continued)



(c)



(d)

Figure 17: Line Graphs of QoS parametric values of four local domains (L_{d1}, L_{d2}, L_{d3}, and L_{d4}) using Gnu-plot. (a): $B_{d(max)}$ for (L_{d1} and L_{d3}) with the proposed algorithm (ISSLM). (b): $B_{d(max)}$ for (L_{d2} and L_{d4}) with the proposed algorithm (ISSLM). (c): $D_{Transfer}$ for (L_{d1} and L_{d3}) with the proposed algorithm (ISSLM). (d): $D_{Transfer}$ for (L_{d2} and L_{d4}) with the proposed algorithm (ISSLM)

5.4 *Case-D: Evaluating the Performance in Terms of QoS Parameters of the Proposed Framework (Logically Distributed Controlled Environment) under ISSLM in Comparison with Traditional Load Balancing Methods*

In Case D, we tested and gathered the QoS parameter values ($B_{d(max)}$, T_h, $D_{transfer}$, $D\_T_{ransmission}\_D_{elay}$) result with the application of the following latest and widely used load balancing techniques mentioned in Table 1 one by one on the user-defined network to compare it with the proposed method (ISSLM) as they also based on Mininet environment as our proposed technique.

- Dynamic Weight Random Selection Method (DWRS) in [50]
- Multiple Regression Based Search Algorithm (MRBS) in [51]
- Adoptive Multiple Objective Load Balancing (AMLOB) in [55]
- Shortest Time Calculation Method (SRT) in [63]
- Conventional Round Robin Method (CRRM) in [64]

The results of Case D are summarized in Table 8. The network QoS parameters are tested for all the above methods, including the proposed technique in two load conditions (normal flow: 200 HTTP requests to be managed by the SDN controller and heavy flow: 20,000 HTTP requests to be managed by the SDN controller). The $D_{elay}\_T_{ransmission}$ is calculated by Eq. (5) in both modes (heavy flow mode: M_2 and normal flow mode: M_1). The differential transmission Delay ($D\_T_{ransmission}\_D_{elay}$), as shown in Eq. (6), is a difference in milliseconds of transmission delay in (Heavy flow mode to normal flow mode: Transmission delay in M_2–Transmission Delay in M_1). The main reason for finding this parameter, in addition to the transmission delay, is to see how much delay is produced in ms when the load shifts from normal to heavy mode. The last column of Table 8 explains the difference in transmission delay in ms when the load shifts from normal mode to heavy flow mode. The QoS metrics ($B_{d(max)}$, T_h, $D_{Transfer}$) obtained under normal loaded conditions include a maximum bandwidth ($B_{d(max)}$) of 4.02 Gb/s and a data transfer ($D_{Transfer}$) volume of 7.02 Gbytes for 15 s.

$$Delay\_Transmission = \frac{[DTransfer(Gbytes)] \times 8}{Bdmax(Gb/s)} \tag{5}$$

$$Overall(D\_Transmission\_Delay) = Delay\_Transmission(M2) - Delay\_Transmission(M1) \tag{6}$$

where (M1 = Normal data traffic mode) and M2 = (Heavy data traffic mode).

Table 8: Contrasting QoS variables results calculated from Iperf utility

Technique utilized	Time in (S_{econds})	$B_{d(max)}$ in (Gb/s)	T_h in (Gb/s)	$D_{Transfer}$ in (G-bytes)	$D\_T_{ransmission}\_D_{elay}$
Research technique ISSLM Algorithm	15	3.469	3.232	6.059	2.80
(SRT)	15	1.499	1.40	2.629	60.58
(CRRM) method	15	1.449	1.386	2.599	379.10
(DWRS) method	15	3.229	3.013	5.649	25.564
(MRBS) method	15	3.025	2.82	5.299	43.78
(AMLOB) method	15	2.796	2.6133	4.899	47.06

Figs. 18–20 show the contrast of QoS variable ($B_{d(max)}$, $D_{Transfer}$, $D\_T_{ransmission}\_D_{elay}$) results obtained from the Iperf utility under the proposed framework (logically distributed) using the Gnu plot.

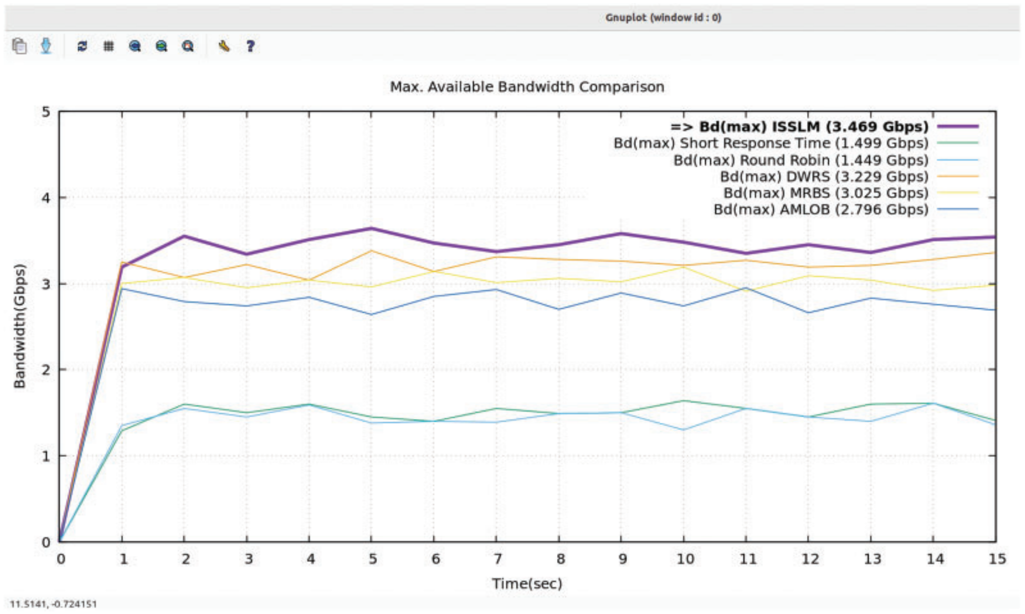


Figure 18: Contrasting the QoS variable ($B_{d(max)}$) results from different traditional methods with the research technique (ISSLM)

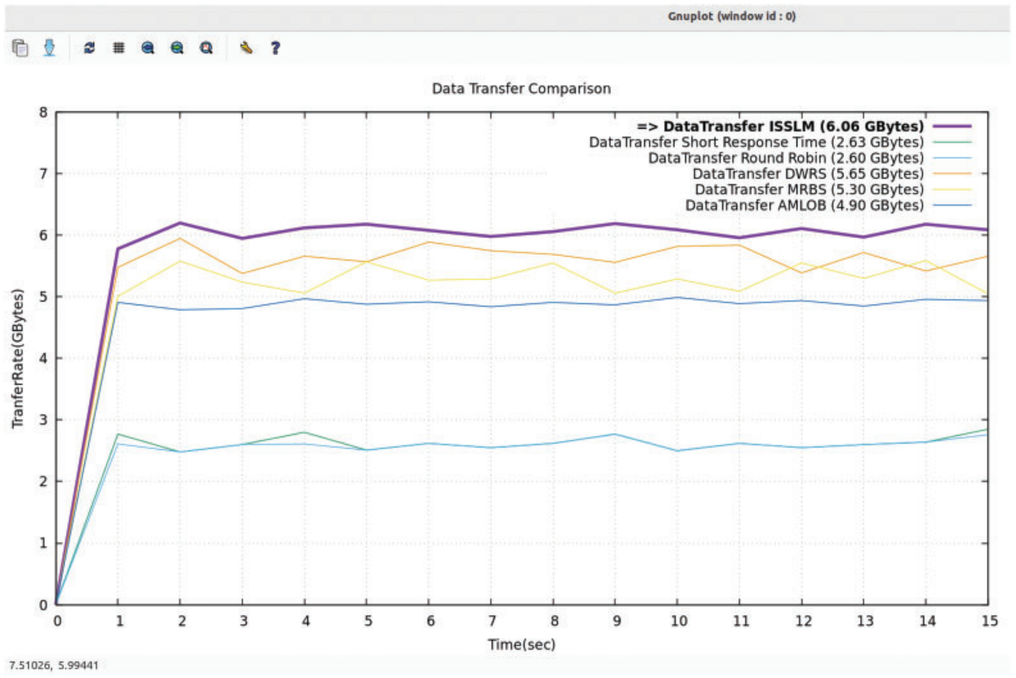


Figure 19: Contrasting QoS variable ($D_{Transfer}$) results from different traditional methods with the research technique (ISSLM)

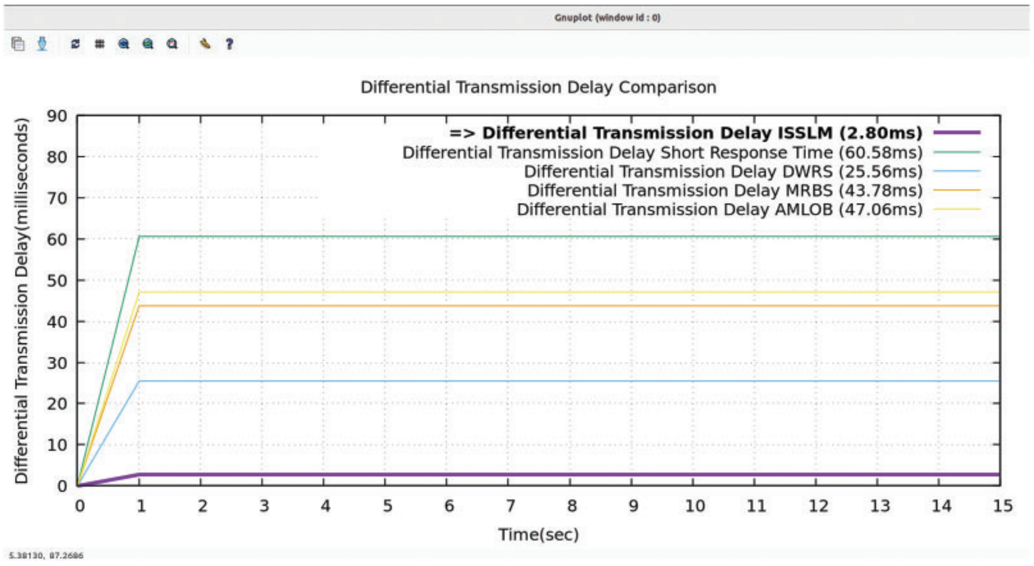


Figure 20: Contrasting QoS variable ($D\_T_{ransmission}\_D_{elay}$) results from different traditional methods with the research technique (ISSLM)

5.4.1 Summarizing QoS Parameters ($B_{d(max)}$, T_h, $D_{Transfer}$, $D\_T_{ransmission}\_D_{elay}$, etc.) Result of Comparative Analysis

When a network is framed in a logically distributed controlled environment, as shown in Fig. 15, the QoS parameters ($B_{d(max),}$ T_h, $D_{Transfer}$, $D\_T_{ransmission}\_D_{elay}$, etc.) results with the proposed algorithm ISSLM are far more superior as compared to QoS parameters obtained from the traditional methods namely: SRT, CRRM, DWRS, MRBS and AMLOB. The B_d value in the selected network with the proposed algorithm-based framework is (3.649 Gb/s) as compared to (1.499, 1.449, 3.229, 3.025, 2.796 Gb/s) for (SRT, CRRM, DWRS, MRBS, and AMLOB), respectively. The T_h value in the selected network with the proposed algorithm-based framework is (3.232 Gb/s) as compared to (1.40, 1.386, 3.013, 2.82, 2.6133 Gb/s) for (SRT, CRRM, DWRS, MRBS, and AMLOB), respectively. The $D_{transfer}$ value in the selected network with the proposed algorithm-based framework is (6.059 Gbytes) as compared to (2.629, 2.599, 5.649, 5.299, 4.899 Gbytes) for (SRT, CRRM, DWRS, MRBS, and AMLOB) respectively and $D\_T_{ransmission}\_D_{elay}$ in the selected network with proposed algorithm based framework is (2.80 ms) as compared to (60.58, 379.10, 25.564, 43.78, 47.06 ms) for (SRT, CRRM, DWRS, MRBS and AMLOB), respectively.

6 Conclusion

In this research article, we have accomplished our goal of magnifying the network quality parameters (maximum available bandwidth, greater throughput, and higher data transfer) in large DCNs by applying the proposed algorithm, ISSLM on SDN controllers arranged in logically distributed arrangement with the utilization of two new methods (1) A large SDN based DCN is divided into sub-networks with independent controllers to overcome the bottleneck issues of the unified controller and also nullifying the compatibilities issues with use of multiple controllers in single large network. (2) Each local controller manages the load by adopting the ISSLM algorithm that performs the desired task of shifting load to the most suitable controller based on three conditions (A-server have less RPS value (OR) B-server with fewer concurrent requests (OR) C-server with less response time). For testing, the virtual network (with many network devices) is formed on Mininet, and simulation is performed in three different configurations: (1) A single POX controller-based SDN controller arrangement designed on Mininet without any server load management algorithm for managing network traffic. (2) A single POX controller-based SDN controller arrangement designed on Mininet with a proposed server load management algorithm (ISSLM) for managing network traffic. (3) Deployment of SDN in logically distributed controlled framework arrangement designed on Mininet with proposed server load management algorithm (ISSLM) for managing network data traffic. Case C's QoS parametrized values ($Bd_{(max)}$, T_h, $D_{Transfer}$, $D\_T_{ransmission}\_D_{elay}$, etc.) showed that the proposed framework with ISSLM algorithm performed better server load management in Large DCNs. The comparative analysis results of the proposed algorithm ISSLM are far more superior as compared to QoS parameters obtained from the traditional methods, namely: SRT, CRRM, DWRS, MRBS and AMLOB. The Bd_{max} value in the selected network with the proposed algorithm-based framework is (3.649 Gb/s) as compared to (1.499, 1.449, 3.229, 3.025, 2.796 Gb/s) for (SRT, CRRM, DWRS, MRBS and AMLOB), respectively. The T_h value in the selected network with the proposed algorithm-based framework is (3.232 Gb/s) as compared to (1.40, 1.386, 3.013, 2.82, 2.6133 Gb/s) for (SRT, CRRM, DWRS, MRBS, and AMLOB), respectively. The Dtransfer value in the selected network with the proposed algorithm based framework is (6.059 Gbytes) as compared to (2.629, 2.599, 5.649, 5.299, 4.899 Gbytes) for (SRT, CRRM, DWRS, MRBS and AMLOB), respectively and $D\_T_{ransmission}\_D_{elay}$ in the selected network with proposed algorithm based framework is (2.80 ms) as compared to (60.58, 379.10, 25.564, 43.78, 47.06 ms) for (SRT, CRRM, DWRS, MRBS and AMLOB), respectively. However, one possible limitation of this research study is that it has been conducted in a controlled or simulated environment; the obtained results can show slight variation when the proposed technique is made functional in real-world conditions and is strongly tied to the performance and reliability of the SDN controllers.

Acknowledgement: Special thanks to Shahid Atiq, Muhammad Majid Hussain, and Intisar Ali Sajjad for guiding and supporting the implementation of the QoS Enhancement Method ISSLM Algorithm. Also, special thanks to Malik Muhammad Abdul Basit for manuscript preparation.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The author's contributions to this paper are as follows: study conception and design: Khawaja Tahir Mehmood, Shahid Atiq; data collection: Khawaja Tahir Mehmood, Shahid Atiq; analysis and interpretation of results: Khawaja Tahir Mehmood, Shahid Atiq; draft manuscript preparation: Khawaja Tahir Mehmood, Shahid Atiq, Intisar Ali Sajjad, Muhammad Majid Hussain, Malik M. Abdul Basit. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The supporting data related to the algorithm and simulation material technical details are with the corresponding author and can be provided upon appropriate request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare that have no conflicts of interest to report regarding the present study.

References

1. Cisco. Annual internet report (2018–2023) white paper. Cisco; 2020. Available from: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. [Accessed 2020].
2. Sloane T. Software-defined networking: the new norm for networks. Open Networking Foundation; 2013. Available from: https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/. [Accessed 2020].
3. Gomez-Rodriguez JR, Sandoval-Arechiga R, Ibarra-Delgado S, Rodriguez-Abdala VI, Vazquez-Avila JL, Parra-Michel R. A survey of software-defined networks-on-chip: motivations, challenges and opportunities. Micromachines. 2021 Feb 12;12(2):183. doi:10.3390/mi12020183.
4. Jenabzadeh MR, Ayatollahitafti V, Mollakhalili MR, Mollahoseini Ardakani MR. VNR_CCP: a new approach to congestion control using virtualization technique and switch migration in SDN. J Model Eng. 2024 May;22(76):85–98.
5. Kaur S, Sandhu AK, Bhandari A. Investigation of application layer DDoS attacks in legacy and software-defined networks: a comprehensive review. Int J Inf Secur. 2023 Aug 7;22(6):1949–88. doi:10.1007/s10207-023-00728-5.
6. Salti IA, Zhang N. An effective, efficient and scalable link discovery (EESLD) framework for hybrid multi-controller SDN networks. IEEE Access. 2023 Jan;11:140660–86. doi:10.1109/ACCESS.2023.3339381.
7. Imran Hussain S, Yesvanth R, Yuvarajapathi V. Implementing OpenFlow, exploring the present and future software-defined networks ecosystem. In: International Conference on Sustainable Communication Networks and Application (ICSCNA), 2023; Theni, India; p. 280–5.
8. Sezer S, Scott-Hayward S, Chouhan P, Fraser B, Lake D, Finnegan J, et al. Are we ready for SDN? Implementation challenges for software-defined networks. IEEE Commun Mag. 2013;51(7):36–43. doi:10.1109/MCOM.2013.6553676.
9. Karakus M, Durresi A. A survey: control plane scalability issues and approaches in Software-Defined Networking (SDN). Comput Netw. 2017 Jan;112:279–93. doi:10.1016/j.comnet.2016.11.017.
10. Javadpour A, Wang G, Rezaei S. Resource management in a peer-to-peer cloud network for IoT. Wirel Pers Commun. 2020 Aug;115(3):2471–88. doi:10.1007/s11277-020-07691-7.
11. Javadpour A, Wang G. cTMvSDN: improving resource management using combination of Markov-process and TDMA in software-defined networking. J Supercomput. 2021;Jul;78(3):3477–99. doi:10.1007/s11227-021-03871-9.
12. Manzoor S, Kayani MA, Ali N, Ratyal NI, Mohamed HG. TiWA: achieving tetra indicator Wi-Fi associations in software defined Wi-Fi networks. IEEE Access. 2023 Jan;11:89520–34. doi:10.1109/ACCESS.2023.3307476.
13. Kalafatidis S, Demiroglou V, Mamatas L, Tsaoussidis V. Experimenting with an SDN-Based NDN deployment over wireless mesh networks. In: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2022; New York, NY, USA; p. 1–6.
14. Kobo HI, Abu-Mahfouz AM, Hancke GP. A survey on software-defined wireless sensor networks: challenges and design requirements. IEEE Access. 2017;5:1872–99. doi:10.1109/ACCESS.2017.2666200.
15. Kreutz D, Ramos FMV, Esteves Verissimo P, Esteve Rothenberg C, Azodolmolky S, Uhlig S. Software-defined networking: a comprehensive survey. Proc IEEE. 2015 Jan;103(1):14–76. doi:10.1109/JPROC.2014.2371999.
16. Shah N, Giaccone P, Rawat DB, Rayes A, Zhao N. Solutions for adopting software defined network in practice. Int J Commun Syst. 2019 May 1;32(17):e3990. doi:10.1002/dac.3990.
17. Zuo QY, Chen M, Zhao GS, Xing CY, Zhang GM, Jiang PC. Research on OpenFlow-based SDN technologies. J Softw. 2013 Dec 1;24(5):1078–97. doi:10.3724/SP.J.1001.2013.04390.

18. Stribling J, Sovran Y, Zhang I, Pretzer X, Li J, Kaashoek MF, et al. Flexible, wide-area storage for distributed systems with WheelFS. Netw Syst Des Implement. 2009 Apr;4:43–58.
19. Koponen T, Casado M, Gude N, Stribling J, Poutievski L, Zhu M, et al. Onix: a distributed control platform for large-scale production networks. Proc OSDI. 2010;10:1–6.
20. Medved J, Varga R, Tkacik A, Gray K. OpenDaylight: towards a model-driven SDN controller architecture. In: Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2014; Sydney, NSW, Australia; p. 1–6.
21. Akka. Build concurrent, distributed, and resilient message-driven applications for java and scala. Available from: https://akka.io/. [Accessed 2020].
22. Suh D, Jang S, Han S, Pack S, Kim T, Kwak JY. On performance of OpenDaylight clustering. In: IEEE NetSoft Conference and Workshops (NetSoft), 2016; Seoul, Republic of Korea; p. 407–10.
23. Berde P, Gerola M, Hart J, Higuchi Y, Kobayashi M, Koide T, et al. ONOS: towards an open, distributed SDN OS. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, 2014; Chicago, IL, USA; p. 1–6.
24. Sakic E, Kellerer W. Response time and availability study of RAFT consensus in distributed SDN control plane. IEEE Trans Netw Serv Manag. 2018 Mar;15(1):304–18. doi:10.1109/TNSM.2017.2775061.
25. Atomix. A reactive java framework for building fault-tolerant distributed systems. Available from: https://atomix.io/ [Accessed 2020].
26. Hunt P, Konar M, Junqueira FP, Reed B. ZooKeeper: wait-free coordination for internet-scale systems. In: USENIX Annual Technical Conference, 2010; Boston, MA, USA; p. 11.
27. Rehman AU, RuiL A, Barraca JP. Fault-tolerance in the scope of software-defined networking (SDN). IEEE Access. 2019;7:124474–90. doi:10.1109/ACCESS.2019.2939115.
28. Katta N, Zhang H, Freedman M, Rexford J. Ravana: controller fault-tolerance in software-defined networking. In: Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, 2015; Santa Clara, CA, USA; p. 1–13.
29. Mantas A, Ramos FMV. Consistent and fault-tolerant SDN with unmodified switches. Cornell University: USA; 2016 Jan.
30. Yeganeh S, Ganjali Y. Kandoo: a framework for efficient and scalable offloading of control applications. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ACM SIGCOMM, 2012; Helsinki, Finland; p. 19–24.
31. Jain S, Zhu M, Zolla J, Hölzle U, Stuart S, Vahdat A, et al. B4: experience with a globally-deployed software defined WAN. ACM SIGCOMM Comput Commun Rev. 2013 Aug;43(4):3–14. doi:10.1145/2486001.2486019.
32. Hong CY, Kandula S, Mahajan R, Zhang M, Gill V, Nanduri M, et al. Achieving high utilization with software-driven WAN. In: Proceedings of the ACM SIGCOMM, 2013 Conference on SIGCOMM, 2013; Hong Kong, China; p. 15–26.
33. Phemius K, Bouet M, Leguay J. DISCO: distributed multi-domain SDN controllers. In: 2014 IEEE Network Operations and Management Symposium (NOMS), 2014; Krakow, Poland.
34. Basavaraju N, Alexander N, Seitz J. Performance evaluation of advanced message queuing protocol (AMQP): an empirical analysis of AMQP online message brokers. In: International Symposium on Networks, Computers and Communications (ISNCC), 2021; Dubai, United Arab Emirates; p. 1–8.
35. Benamrane F, Ben Mamoun M, Benaini R. An East-West interface for distributed SDN control plane: implementation and evaluation. Comput Electr Eng. 2017 Jan;57:162–75. doi:10.1016/j.compeleceng.2016.09.012.
36. Esteve Rothenberg C, Trindade Nascimento M, Teixeira Godoy H, Carlos S, Raszuk R. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In: Proceedings of the First

Workshop on Hot Topics in Software Defined Networks, ACM SIGCOMM, 2012; Helsinki, Finland; p. 13–8.

37. Santos MAS, Nunes BAA, Obraczka K, Turletti T, de Oliveira BT, Margi CB. Decentralizing SDN's control plane. In: 39th Annual IEEE Conference on Local Computer Networks, 2014; Edmonton, AB, Canada; p. 402–5.
38. Stringer J, Pemberton D, Fu Q, Lorier C, Nelson RE, Bailey JJ, et al. Cardigan: SDN distributed routing fabric going live at an Internet exchange. In: 2014 IEEE Symposium on Computers and Communications (ISCC), 2014; Funchal, Portugal; p. 1–7.
39. Lapeyrade R, Bruyère M, Owezarski P. OpenFlow-based migration and management of the TouIX IXP. In: IEEE/IFIP Network Operations and Management Symposium, 2016; Istanbul, Turkey; p. 1131–6.
40. Malboubi M, Wang L, Chuah CN, Sharma P. Intelligent SDN based traffic (de)aggregation and measurement paradigm (iSTAMP). In: IEEE INFOCOM 2014—IEEE Conference on Computer Communications, 2014; Toronto, ON, Canada; p. 934–42.
41. Endeavour. Project. Available from: https://www.h2020-endeavour.eu/. [Accessed 2024].
42. Morgan H. Atlantic wave-SDX: a distributed intercontinental experimental software defined exchange for research and education networking; 2015. Available from: https://itnews.fiu.edu/wp-content/uploads/sites/8/2015/04/AtlanticWaveSDX-Press-Release_FinalDraft.pdf [Accessed 2020].
43. Lin P, Bi J, Wolff S, Wang Y, Xu A, Chen Z, et al. A west-east bridge based SDN inter-domain testbed. IEEE Commun Mag. 2015 Feb;53(2):190–7. doi:10.1109/MCOM.2015.7045408.
44. Mehmood KT, Atiq S, Hashmi MW. Predictive analysis of telecom system quality parameters with SDN (Software Define Networking) controlled environment. Elem Educ Online. 2021 Jan;20(3):2342–55.
45. Advanced Layer 2 System, Internet2. Ann Arbor, MI, USA: Advanced Technology Community Founded by the U.S. Research and Education Communit; Dec. 2014. Available from: https://www.internet2.edu/products-%20services/advanced-networking/layer-2-services/. [Accessed 2024].
46. Kotronis V, Gämperli A, Dimitropoulos X. Routing centralization across domains via SDN: a model and emulation framework for BGP evolution. Comput Netw. 2015 Dec;92:227–39. doi:10.1016/j.comnet.2015.07.015.
47. Yu M, Rexford J, Freedman MJ, Wang J. Scalable flow-based networking with DIFANE. In: SIGCOMM'10: Proceedings of the ACM SIGCOMM, 2010; New Delhi, India; p. 351–62.
48. Aly WHF. Generic controller adaptive load balancing (GCALB) for SDN networks. J Comput Netw Commun. 2019 Dec;2019:1–9. doi:10.1145/1851182.1851224.
49. Mehmood KT, Atiq S, Hussain MM. Enhancing QoS of telecom networks through server load management in software-defined networking (SDN). Sensors. 2023 Jan;23(23):9324. doi:10.3390/s23239324.
50. Chiang ML, Cheng HS, Liu HY, Chiang CY. SDN-based server clusters with dynamic load balancing and performance improvement. Cluster Comput. 2020 May;24(1):537–58. doi:10.1007/s10586-020-03135-w.
51. Begam GS, Sangeetha M, Shanker NR. Load balancing in DCN servers through SDN machine learning algorithm. Arab J Sci Eng. 2021 Jul;47(2):1423–34. doi:10.21203/rs.3.rs-277161/v1.
52. Malbašić T, Bojović PD, Bojović Ž., Šuh J, Vujošević D. Hybrid SDN networks: a multi-parameter server load balancing scheme. J Netw Syst Manag. 2022 Jan;30(2):1–29. doi:10.21203/rs.3.rs-383737/v1.
53. Liang S, Jiang W, Zhao F, Zhao F. Load balancing algorithm of controller based on SDN architecture under machine learning. J Syst Sci Inform. 2020 Dec;8(6):578–88. doi:10.21078/JSSI-2020-578-11.
54. Ahmad S, Jamil F, Ali A, Khan E, Ibrahim M, Keun Whangbo T. Effectively handling network congestion and load balancing in software-defined networking. Comput Mater Contin. 2022;70(1):1363–79. doi:10.32604/cmc.2022.017715.
55. Saxena MC, Sabharwal M, Bajaj P. Review of SDN-based load-balancing methods, issues, challenges, and roadmap. Int J Electr Comput Eng Syst. 2023 Nov;14(9):1031–49. doi:10.32985/ijeces.14.9.8.

56. Haidi AM, Au TW, Shah H. Single cluster load balancing using SDN: performance comparison between floodlight and POX. In: IEEE 19th International Conference on Communication Technology (ICCT), 2019; Xi'an, China.

57. Ejaz S, Iqbal Z, Azmat Shah P, Bukhari BH, Ali A, Aadil F. Traffic load balancing using software defined networking (SDN) controller as virtualized network function. IEEE Access. 2019;7:46646–58. doi:10.1109/ACCESS.2019.2909356.

58. Gasmelseed H, Ramar R. Traffic pattern-based load-balancing algorithm in software-defined network using distributed controllers. Int J Commun Syst. 2018 Nov;32(7):e3841. doi:10.1002/dac.3841.

59. Xu Y, Cello M, Wang IC, Walid A, Wilfong G, Wen CHP, et al. Dynamic switch migration in distributed software-defined networks to achieve controller load balance. IEEE J Sel Areas Commun. 2019 Mar;37(3):515–29. doi:10.1109/JSAC.2019.2894237.

60. 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 Jan;6:73774–85. doi:10.1109/ACCESS.2018.2884708.

61. Vyakaranal SB, Naragund JG. Weighted round-robin load balancing algorithm for software-defined network. In: Emerging Research in Electronics, Computer Science and Technology, 2019; Singapore; p. 375–87.

62. Sathyanarayana S, Moh M. Joint route-server load balancing in software defined networks using ant colony optimization. In: International Conference on High Performance Computing and Simulation, 2016; Innsbruck, Austria; p. 156–63.

63. Zhong H, Fang Y, Cui J. LBBSRT: an efficient SDN load balancing scheme based on server response time. Future Gener Comput Syst. 2017 Mar;68:183–90. doi:10.1016/j.future.2016.10.001.

64. Hamed MI, ElHalawany BM, Fouda MM, Eldien AST. Performance analysis of applying load balancing strategies on different SDN environments. Benha J Appl Sci. 2017 Mar;2(1):91–7. doi:10.21608/bjas.2017.163983.

65. Hai NT, Kim DS. Efficient load balancing for multi-controller in SDN-based mission-critical networks. In: IEEE 14th International Conference on Industrial Informatics (INDIN), 2016; Poitiers, France; p. 420–5.