computers

Article

A Novel Dynamic Software-Defined Networking Approach to Neutralize Traffic Burst

Aakanksha Sharma [1,2,*], Venki Balasubramanian [1,3] and Joarder Kamruzzaman [1,3]

[1] Institute of Innovation, Science and Sustainability, Federation University Australia, Ballarat, VIC 3350, Australia; v.balasubramanian@federation.edu.au (V.B.); joarder.kamruzzaman@federation.edu.au (J.K.)
[2] Melbourne Institute of Technology (MIT), Melbourne, VIC 3000, Australia
[3] Centre for Smart Analytics (CSA), Federation University Australia, Ballarat, VIC 3350, Australia
* Correspondence: aakanksha.sharma2@federation.edu.au

Abstract: Software-defined networks (SDN) has a holistic view of the network. It is highly suitable for handling dynamic loads in the traditional network with a minimal update in the network infrastructure. However, the standard SDN architecture control plane has been designed for single or multiple distributed SDN controllers facing severe bottleneck issues. Our initial research created a reference model for the traditional network, using the standard SDN (referred to as SDN hereafter) in a network simulator called NetSim. Based on the network traffic, the reference models consisted of light, modest and heavy networks depending on the number of connected IoT devices. Furthermore, a priority scheduling and congestion control algorithm is proposed in the standard SDN, named extended SDN (eSDN), which minimises congestion and performs better than the standard SDN. However, the enhancement was suitable only for the small-scale network because, in a large-scale network, the eSDN does not support dynamic SDN controller mapping. Often, the same SDN controller gets overloaded, leading to a single point of failure. Our literature review shows that most proposed solutions are based on static SDN controller deployment without considering flow fluctuations and traffic bursts that lead to a lack of load balancing among the SDN controllers in real-time, eventually increasing the network latency. Therefore, to maintain the Quality of Service (QoS) in the network, it becomes imperative for the static SDN controller to neutralise the on-the-fly traffic burst. Thus, our novel dynamic controller mapping algorithm with multiple-controller placement in the SDN is critical to solving the identified issues. In dSDN, the SDN controllers are mapped dynamically with the load fluctuation. If any SDN controller reaches its maximum threshold, the rest of the traffic will be diverted to another controller, significantly reducing delay and enhancing the overall performance. Our technique considers the latency and load fluctuation in the network and manages the situations where static mapping is ineffective in dealing with the dynamic flow variation.

Keywords: static and dynamic controller mapping; flow fluctuation; latency minimization; load balancing

check for updates

Citation: Sharma, A.; Balasubramanian, V.; Kamruzzaman, J. A Novel Dynamic Software-Defined Networking Approach to Neutralize Traffic Burst. *Computers* **2023**, *12*, 131. https://doi.org/10.3390/computers12070131

Academic Editor: Paolo Bellavista

Received: 12 May 2023
Revised: 16 June 2023
Accepted: 26 June 2023
Published: 27 June 2023

1. Introduction

The global advancement of the Internet of Things (IoT) has poised traditional network traffic for explosive growth. The prediction in the literature shows that in the future, trillions of smart devices will connect to transfer useful information [1]. Accommodating such proliferation of devices in the traditional network infrastructure is a significant challenge due to the absence of centralised control, making it tedious to implement device management and network protocol updates. Expanding the traditional infrastructure and manual traffic control methods is not enough to cope with the exponential growth of IoT devices.

SDN technology manages the network devices and provides a complete global view of the network to configure, monitor, and troubleshoot quickly. SDN decouples the data and control plane, making the network control directly programmable as it is decoupled from forwarding functions. Thus, SDN is programmable, while traditional networks are non-programmable as their data and control planes are embedded in the same device. Troubleshooting in SDN networks is easy, based on a centralised SDN controller. Still, traditional networks are challenging to troubleshoot due to the static deployment of hardware devices and their heterogeneity issues. These SDN benefits make it capable of overcoming the problems of traditional networks. The literature demonstrates that SDN can also handle bandwidth starvation issues by allocating bandwidth more efficiently due to its directly programmable, agile, and centrally managed network control. Moreover, it significantly increases the speed and network control flexibility compared to traditional networks.

The current SDN architecture of the control plane has been designed for a single controller or multiple distributed controllers; however, a large-scale network with a logically centralised single controller faces severe bottleneck issues. Our research was initiated by creating a reference model by placing the standard SDN in a traditional network and analysing how it overcomes the traditional network issues that enhance network performance. Implementing SDN in a traditional network reduces network delays and jitters while improving the network throughput. Subsequently, overall network performance for all network traffic types was enhanced. However, a huge network delay is still noted in heavy network traffic. Our research identified that the gap is due to a lack of packet priority processing, congestion control mechanisms, and an unbalanced network load when the load increases exponentially, which increases packet loss and network delay.

To overcome the identified issues, real-time applications should be appropriately prioritised to improve network performance. For example, reliable and timely data delivery is necessary for critical applications such as vehicular networks and healthcare monitoring systems. Therefore, the IoT heterogeneous network must maintain the network QoS by enabling packet prioritisation and congestion control mechanisms.

Thus, we proposed extended SDN (eSDN) using the standard SDN's packet priority processing and congestion control algorithm. The standard SDN is optimised by minimising congestion to achieve eSDN, which can accommodate the growing number of heterogeneous IoT devices. The simulation analyses of the reference network show that eSDN performs better than the standard SDN for all network traffic by processing packets per their priorities, minimising packet loss and overall network delay. Therefore, a significant improvement in network performance is achieved by implementing eSDN in the standard SDN environment for mild and moderate traffic. However, heavy traffic still has extensive network delays due to the static SDN controller deployment in eSDN. When the SDN switch observes a traffic burst in eSDN, the corresponding SDN controller might often become overloaded. A scalable and trusted platform is needed to manage the numerous connected IoT devices and efficient network technology to support dynamic network management. So, the flow-based dynamic controller mapping is significant for maintaining the SDN's optimal network management by improving the network QoS.

Our further research improves the network QoS by maintaining the network load with dynamic mapping among SDN controllers when traffic bursts. Thus, it proposes a multiple SDN controller placement in the network with a dynamic mapping algorithm that minimizes the network delay and jitter. The approach maps the SDN controllers dynamically according to the load fluctuations.

Initially, we simulated the traditional networks using the standard SDN from NetSim Simulator to identify the delay, jitter, and throughput deficiencies for heterogeneous IoT devices. Then, we proposed and implemented the eSDN in the standard SDN to improve network performance. This proposed method prioritises processing urgent information, reducing delays and improving the network's QoS. However, it leads to only one SDN controller being overloaded, thus affecting the network performance. Therefore, we proposed dSDN for the dynamic controller mapping that minimises delay and jitter.

Overall, the presented work in this paper makes the following contributions:

1. SDN-based new load balancing techniques are introduced, termed eSDN and dSDN,
   (a) eSDN provides packet priority processing and supports congestion control, which is better than the standard SDN-based solution.
   (b) dSDN can deal with imbalanced network load due to flow fluctuation significantly through the dynamic mapping of controllers.
2. The proposed solution is scalable to large-scale networks and will allow network providers to manage heavy traffic from heterogeneous sources, such as growing streaming and IoT applications."
3. Extensive simulation results with varied types of network traffic demonstrate that dSDN outperforms eSDN and traditional SDN in terms of throughput, delay, and jitter and ensures better QoS.

The rest of the paper is organised as follows: Section 2 summarises related works in the current literature; Section 3 introduces standard SDN in traditional networks; the problem formulation, system model and proposed methods are mentioned in Section 4; network setup is introduced in Section 5; and Section 6 presents the simulation of IoT networks. The overall comparison of the results is discussed in Section 7, followed by a conclusion in Section 8.

2. Related Works

Our literature considers several survey articles on SDN's architecture, such as [2–9]. The studies in [5–7,10] reviewed the SDN while introducing programmable networks that considered controllers and simulators to simulate the architecture. Karakus et al. [11] revealed that SDN has emerged in response to traditional network limitations. Still, scalability is an issue in the SDN network. The scalability problem is identified as the controller's scalability issues and summarised in the discussion into topology and mechanisms-based scalability issues. The potential challenges that need to be considered for future research include controller failure, state/policy distribution/consistency, flow rule setup latency, and SDN controller placement problems.

SDN provides various technical benefits, particularly network traffic engineering [12], due to SDN's separation of data and control plane. However, separating the control and forwarding planes in SDN results in issues such as controller placement [13] and network management problems due to dynamic flow [14]. A single controller is insufficient for dynamic network management in large-scale networks. Therefore, the multiple distributed SDN controller architecture is deployed to manage the network [15–17].

However, due to an imbalance of load distribution among the SDN controllers, most controllers are underutilized [18,19]. As the network grows, its data increases; thus, if controllers are overloaded, the response time increases. If the number of requests exceeds the controller's processing power, requests may get lost. Therefore, these approaches establish a static mapping between the SDN switch and controllers, causing an imbalanced load in the network and decreasing performance with the dynamic network traffic flow. To ensure network QoS, networks must be programmable; for example, the flows, which have high priority or are delay-sensitive, must be forwarded through the shortest path. The healthcare data cannot be delayed in the case of remote patient monitoring data. Low-sensitive flows can be forwarded using reliable paths but may be the longest paths with minimum loss.

In recent years, increased efforts have been invested in centralised flow control methods based on SDN [20,21]. Kim et al. [22] suggested that SDN-based technologies must be applied for network load balancing, traffic forwarding, and better bandwidth utilisation. A centralised method for flow management using a network operating system named NOX is presented in [23,24]. It manipulates the switches according to the management decisions. If the incoming packet at a switch matches a flow entry, the switch applies the related actions. A round-robin load-balancing method, which uses a circular queue to decide where to send the request of each incoming client, is proposed by [25] and responds to DNS requests with a list of IP addresses. However, these approaches need to be revised to handle real-time traffic fluctuation. For instance, the work in [26] is based on load shifting for a data centre network when the flow is at its peak. However, this shifting method is not suitable for centralised networks. Moreover, centralised rerouting decisions are essential in most mechanisms, but the cost of rerouting negatively affects their decision-making efficiency.

However, the present centralised SDN cannot handle the IoT dynamic requirements. As a result, available resources are underutilized in centralised SDN due to the lack of a dynamic rule-placement approach. Thus, an efficient approach is needed as billions more devices will be connected in the future, generating data exponentially [27,28] mentioned that network management is essential to managing the massive collection of information and devices to process the generated data efficiently. The overall network performance depends on resource utilisation; thus, under or over-utilising network components degrades the network performance and minimizes the network utility. So, we need suitable technologies to control the network traffic flows for load balancing and latency minimization.

To balance the network load, better architectures must be designed to enhance the network scalability without overloading the SDN controllers. The main aim is to reduce the central component load without reducing the load balance efficiency. Related to this identified issue, Neghabi et al. [29] described that few mechanisms can work to balance the network load. For example, the SDN switch's default controller can be changed by sending all the requests from the switch to a new controller or associating every switch with more than one controller. Thus, enabling the SDN switch to send some requests to one controller and the rest of the data to another leads to flow distribution; still, this mechanism cannot cope with heavy traffic with flow variation. For a large-scale network, an effective load-balancing algorithm might be required to increase the flexibility of the network.

A static deployment of SDN controllers in [30] uses multiple SDN controllers where more than one SDN controller participates and works cooperatively in the forwarding directions to create various domains. When multiple SDN controllers are randomly placed at different locations, they enhance the network QoS by monitoring and controlling the SDN switches in every environment [30]. Due to this approach, overall network performance increased significantly with effective load distribution among different controllers. Still, controllers cannot handle real-time flow fluctuation due to their static deployment.

Li et al. [31] focused on the zoning problem, dividing the network into zones and letting slave controllers manage the zones separately and synchronize with the master controller for distributed network optimisation. With the help of meta-heuristic approaches, researchers balanced the processing load. Few works implemented the dynamic approach, but those approaches are less effective and only provide restricted customisation flexibility, being not suitable for the proliferation of IoT devices because multiple controllers do not work effectively when deployed due to the flow fluctuation in the network. A comprehensive review of several optimised controller placement problem algorithms in SDN is defined in [9], which raised many research challenges, such as unbalanced network load and computing the optimal number of controllers needed in the network. Network static load balancing is discussed in [32,33], which fails to handle real-time flow fluctuation. Pham et al. [34] illustrate the impact of service chains on the control plane and formulate the dynamic controller/switch mapping (DCSM) problem in Network Function Virtualization (NFV) networks to reduce the operational cost. The experimental results show a reduction in operational cost by up to 31.7% and 28.3% compared to K-Mean and the matching game-based approaches, respectively.

In summary, the existing research only found better solutions for static load balancing and targets only one or two issues related to SDN controllers. No solution exists in the literature that satisfies the network performance, load balancing, latency minimisation, and dynamic flow fluctuations. Therefore, it is imperative to consider dynamic traffic management in the existing network infrastructure by dynamic SDN switch-controller mapping to avoid an unbalanced network load.

Therefore, addressing these challenges is needed to improve the network's scalability and robustness. We proposed a dynamic solution for an unbalanced network load due to flow fluctuation. The dSDN considers dynamic load fluctuation with the dynamic mapping of controllers; none of the SDN controllers goes unbalanced. It enhances the network throughput, minimises the network latency, and enhances the overall network QoS. In this paper, we have proposed and implemented the eSDN and dSDN.

3. Introduction to Standard SDN in Traditional Network Architecture

SDN technology manages the network devices and provides a complete global view of the network to configure, monitor, and troubleshoot quickly. It has two main features: decoupling the control and data planes; and control plane programmability [35]. With the separation of the network from the hardware, policies are no longer to be executed on the hardware itself. SDN enables dynamic and efficient network configuration and centralises network intelligence by separating the infrastructure layer, including network packets (or data), the control layer, and the routing process (or control planes). The control plane is the brain of the SDN network, and it contains all the intelligence because of one or more controllers present in it.

Figure 1 shows the SDN architecture, and its details are mentioned below:

- Infrastructure layer: The bottom layer contains various network devices such as switches, routers, and access points. The SDN controller is responsible for managing and accessing them.
- Application program interfaces (APIs): APIs are the central control point for every network component. SDN controller uses northbound and southbound application program interfaces to interact with the application and infrastructure layer. A southbound interface allows sending commands from a higher-level network component to a lower level. Conversely, the northbound interface allows a lower-level network component to establish communication with a higher level.
- Control layer: The control plane provides the control functionalities for network supervision. It has a centralised SDN controller responsible for communication between network applications and controllers. The SDN controller interacts with the upper and lower application layers using northbound and southbound interface protocols.
- Application layer: It has one or more end-user applications, such as security, load balancer and firewalls. It uses the northbound interface to interact with the SDN controller, and they get an abstract network view by interacting with controllers for internal decision-making processes.

SDN architecture uses the OpenFlow protocol (OFP), which defines the communication among SDN controllers and network devices. The OFP works on the TCP protocol, which establishes the IP connection among the controller and switches or any network device. It includes three essential components defined below [36,37].

- Switches: It uses flow tables containing the action fields associated with every flow entry 25 communication. It evaluates the incoming packet, checks the matching flow table entry, and performs the required action. The switch communicates through the channels initiated when it sends a hello message and some information about the highest OpenFlow (OF) version it supports to the controller; the controller replies to the message and its highest supported OF version. Then switch negotiates on the highest OF level, which they both support. Adding more channels will boost the network connection speed.
- Flow Entries: In the flow table, a flow entry is an element that is used to match and process the packets. When a packet matches a flow entry, the OF switch updates the action set and matches the packets in the flow table to send to the respective ports.
- Controllers: Generally, a controller manages the communication between two or more devices. A physical/virtual device manages traffic forwarding by communicating

with the network devices. It is used to modify, manage, or navigate the data flow among entities like a video game controller that acts as an input device for playing games. The OFP [36] used in SDN is the standard for establishing a communication protocol among SDN controllers and networking devices. It provides various benefits, including programmability, centralised intelligence, and abstraction.

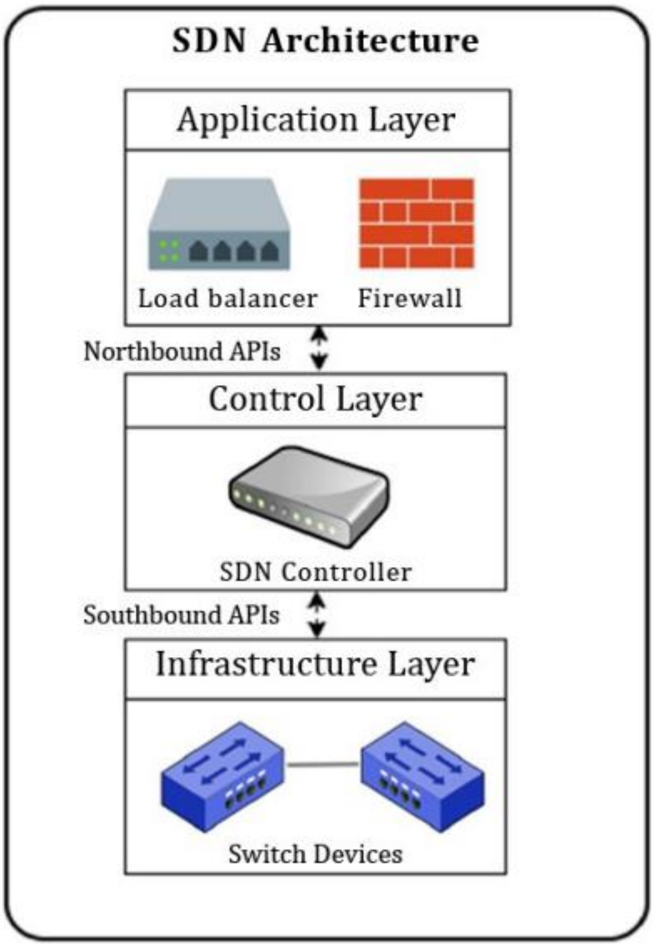


Figure 1. SDN Architecture.

The significant advantage of SDN technology is that it can quickly adapt to new policies using the software because its centralised and distributed approaches provide better results than traditional hardware-based networks. Thus, SDN is more flexible and takes less time to manage the network. Troubleshooting in SDN networks is easy, based on a centralised controller. Still, traditional networks are challenging to troubleshoot due to the static deployment of hardware devices and their heterogeneity issues.

These SDN benefits make it capable of overcoming the issues of traditional networks. Initially, we identify how standard SDN performs compared to the traditional networks by using a standard network simulator, NetSim, to systematically identify the traditional network issues and overcome the issues using standard SDN. The reference models are later used to study the performance of our contributions, eSDN and dSDN. Our simulation results using SDN in the traditional network show that SDN accommodates the need for growing heterogeneous IoT devices. The simulation results also revealed that apart from unbalanced network load and controller placement problems, the SDN also suffers from packet prioritisation and congestion control techniques leading to packet loss and delay. If the packets are processed based on their priorities, for instance, real-time packets such as voice and video get high priority compared to FTP packets, it can enhance the overall

QoS in the network. The section below introduces the priority scheduling and congestion control algorithm (eSDN).

4. Problem Formulation, System Model and Proposed Methods

4.1. Problem Formulation and System Model

The 'Related Work' section identified the traditional network issues with increased Internet traffic. These challenges can be alleviated through the use of SDN. We created our reference model for traditional network architecture and then used SDN to verify the results. The results verified that SDN behaves better than traditional networks by improving the QoS. However, for heavily crowded networks, standard SDN suffers from severe bottleneck issues and huge network delays are noted. The identified research gap is the lack of packet priority processing, congestion control method and multi-controller placement. Another problem is the inability of the SDN to deal with heavy traffic and how the flow can the distributed among the controllers to achieve load balancing.

A scalable and trusted platform is needed to manage the numerous connected IoT devices and efficient network technology to support dynamic network management. So, the flow-based dynamic controller mapping is significant for maintaining the SDN's optimal network management by improving the network QoS. Most of the literature is based on static controller deployment. Few authors focused on the dynamic placement of controllers without considering the dynamic load fluctuation and network latency.

Figure 2 shows an SDN system model on which we built our proposed solutions. Below we propose two methods to address the research problem stated above.

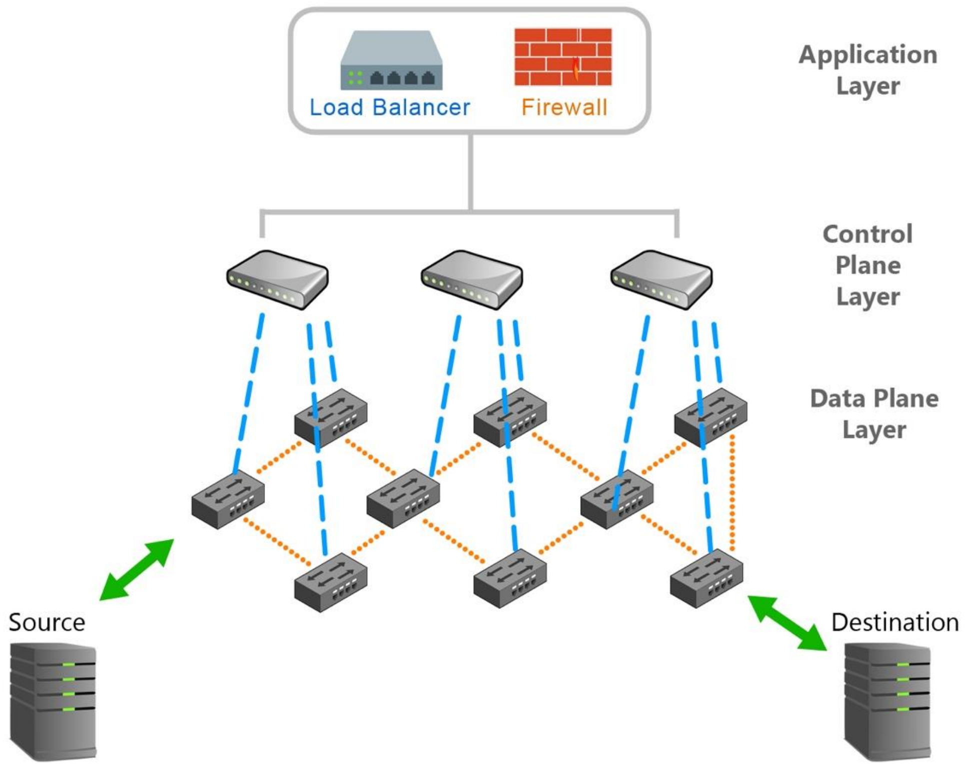


Figure 2. SDN-based System Model.

4.2. Introduction to Priority Scheduling and Congestion Control Algorithm (eSDN) in Standard SDN

Effective resource allocation can enhance the network QoS by considering packet buffering before the transmission. The packet buffering matches the data transmission speed between two devices by managing the queuing mechanisms used for packet transmission and dropping the packets based on bandwidth allocation and buffer space. However, the latency or delay of a packet depends on how long a packet waits before transmission; therefore, the queuing method affects packet latency. Thus, our proposed solution implements packet priority processing with congestion control in standard SDN.

The proposed solution is implemented in a standard SDN controller. Figure 3 is the block diagram of eSDN, and Table 1 is the algorithmic notations showing the commonly used notations and variables. As per our approach, packet priority and type are initialized based on the abovementioned priority table, and then it initializes the packet priority functions. The ADD and GET functions are performed based on the assigned packet priority; if ADD, all packets will be added to the buffer based on their priorities; otherwise, packets will be retrieved per priority. It is essential in some situations where packets from multiple directions arrive at an SDN router. The priority processing allows the SDN router to set the priority levels for all packets from different directions and helps to deliver the packets as per their importance. A congestion control method is introduced with packet priority processing, which works in four phases: slow start, congestion avoidance, fast retransmit, and fast recovery.

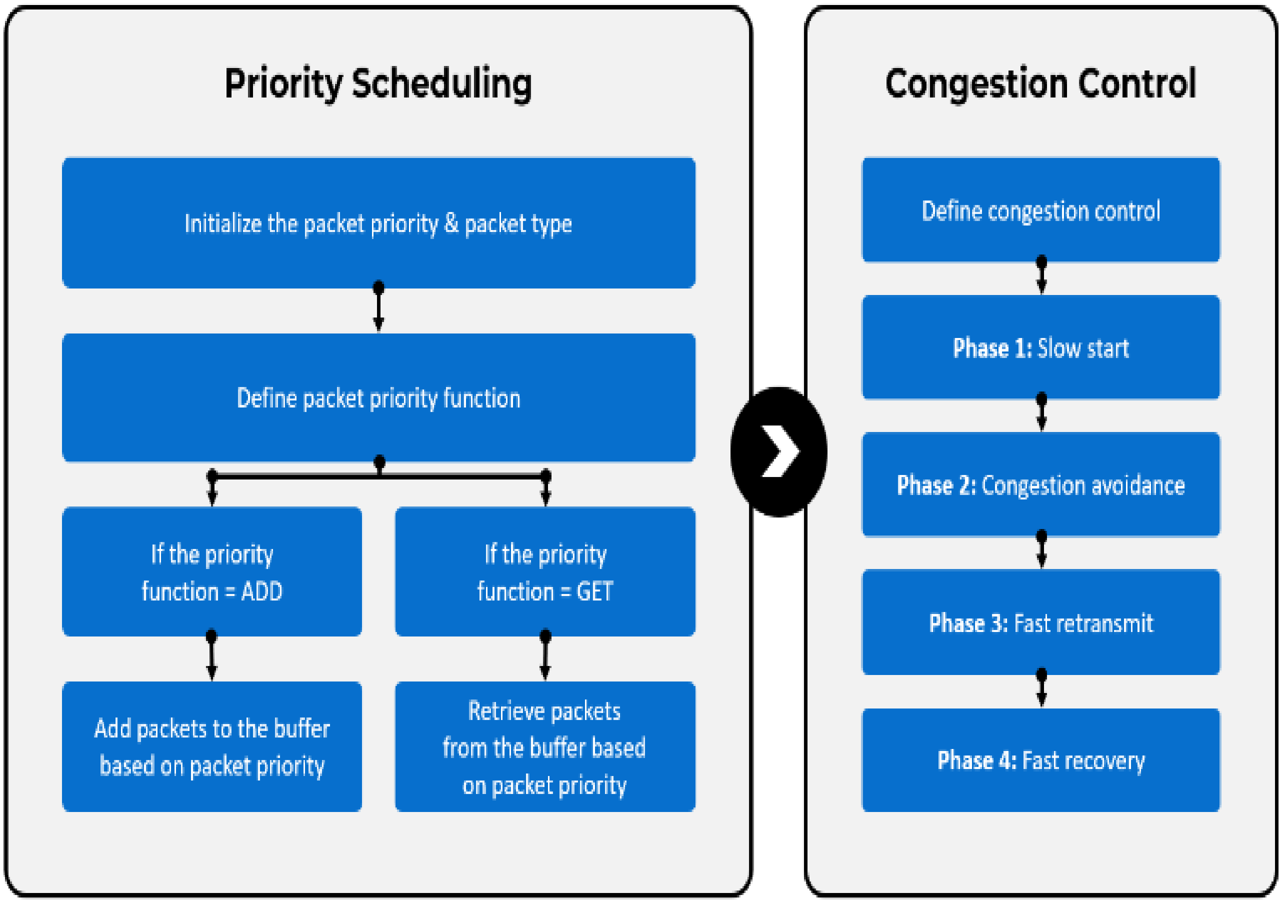

Priority Scheduling
Initialize the packet priority & packet type
Define packet priority function
If the priority function = ADD
If the priority function = GET
Add packets to the buffer based on packet priority
Retrieve packets from the buffer based on packet priority
Congestion Control
Define congestion control
Phase 1: Slow start
Phase 2: Congestion avoidance
Phase 3: Fast retransmit
Phase 4: Fast recovery

Figure 3. Block diagram for eSDN.

The flow chart for our proposed eSDN algorithm is shown in Figure 4. The initial step is to define the packet priorities and packet functions. The ADD and GET functions are performed based on the assigned packet priority. The network congestion is managed by the transmission control protocol (TCP). TCP protocol uses a congestion window and a congestion policy that avoids congestion. The key design principle is avoiding network congestion by adjusting the TCP receiver window of the ACK packet at the SDN controller to reduce the sender's transmission rate. The receiver determines the window size by withholding the acknowledgement of packets sent by the sender.

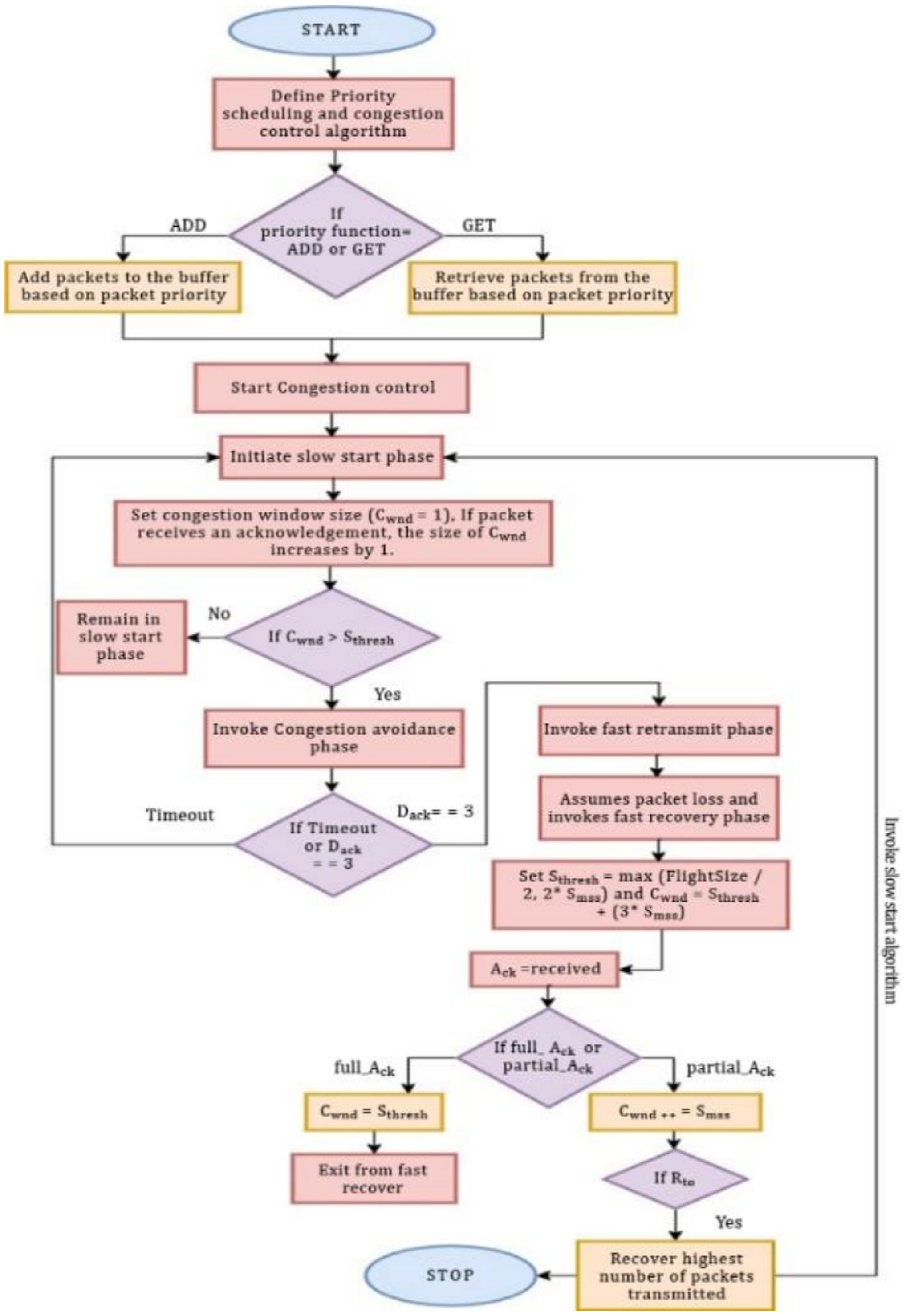

START
Define Priority scheduling and congestion control algorithm
If priority function= ADD or GET
ADD
GET
Add packets to the buffer based on packet priority
Retrieve packets from the buffer based on packet priority
Start Congestion control
Initiate slow start phase
Set congestion window size (C_{wnd} = 1), If packet receives an acknowledgement, the size of C_{wnd} increases by 1.
If $C_{wnd} > S_{thresh}$
No
Remain in slow start phase
Yes
Invoke Congestion avoidance phase
If Timeout or D_{ack} = = 3
Timeout
D_{ack}= = 3
Invoke fast retransmit phase
Assumes packet loss and invokes fast recovery phase
Set S_{thresh} = max (FlightSize / 2, 2* S_{mss}) and C_{wnd} = S_{thresh} + (3* S_{mss})
A_{ck} =received
If full_ A_{ck} or partial_A_{ck}
full_A_{ck}
partial_A_{ck}
C_{wnd} = S_{thresh}
C_{wnd} ++ = S_{mss}
Exit from fast recover
If R_{to}
Yes
Recover highest number of packets transmitted
STOP
Invoke slow start algorithm

Figure 4. Flowchart for eSDN.

The initial phase is a slow start; it updates the congestion window. The congestion window determines the number of packets the sender can send. The initial congestion window size for data transmission is 1; thus, TCP can send one packet until it receives an acknowledgement (ACK). When the sender receives the ACK, the congestion window size increases to two, and the sender can send two packets. After receiving every new ACK, the sender congestion window rises exponentially. In congestion avoidance, the TCP resumes the slow-start phase until it reaches a certain threshold or a packet loss occurs.

Table 1. Notations of Algorithmic Pseudocode for eSDN.

Notations	Descriptions
Pn	Priority function
Pp	Packet priority
Pt	Single packet
Pa	All packets
Ack	Packet acknowledgement
Cwnd	Congestion window size
Smss	Sender maximum segment size
Rmss	Receiver maximum segment size
Sthresh	Slow start threshold value
Dack	Duplicate packet acknowledgement
Rto	Retransmit timeout
Bp	Packet buffer

The next phase it enters is fast retransmitting, which arises when packet loss is observed in the network and invokes the fast recovery phase. The congestion window will increase linearly from 'k' to 'k + 1' after receiving 'k' new ACKs. Due to the observed packet loss, the growth rate in the congestion window slows down.

The fast retransmit improves congestion avoidance with the fast-retransmit method as it considers that duplicate ACKs could signal packet loss in the network. After obtaining duplicate ACKs, if it receives three Dack, the fast-retransmit phase assumes packet loss and invokes the fast recovery phase. The fast recovery is less aggressive than fast retransmitting in reducing congestion windows. It implements the fast recovery method; when three duplicate acknowledgements arrive, the congestion window drops to one, meaning less network congestion and the congestion window need not be highly reduced.

Algorithmic Pseudocode for eSDN

The proposed algorithm that accomplishes our priority scheduling and congestion control algorithm is shown in Algorithm 1.

Algorithm 1 Algorithmic Pseudocode for eSDN

```
1: Initialize Pn
2: if Pn is added, then
3:         for each packet Pt in Pa from high to low priority, do
4:             Add Pt to Buffer packet
5:         end for
6: else Pn gets then
7:         for each packet Pt in Bp from high to low priority, do
8:             Retrieve Pt from the Buffer packet
9:         end for
10: end if
11: Initialize Cwnd = 1, Smss = 1, Sthresh = 64 KB
12: Send Smss
13: while sending the segment, do
14:         if Timeout, then
15:             Set Sthresh = $\underline{Cwnd}$, Cwnd = 1, Smss = 1
16:             Send Smss waiting for ack
17:         else Dack == 3 then
18:             Invoke fast retransmit
```

```
19:             Invoke fast recovery
20:             $Sthresh = \max(\underline{FlightSize}, 2 * Smss)$
21:             $Cwnd = Sthresh + (3 * Smss)$
22:             **if** full Acknowledgement is received, **then**
23:                $Cwnd = Sthresh$
24:                Exit from fast recovery
25:             **else** partial Acknowledgement is received, **then**
26:                $Cwnd = Cwnd + Smss$
27:                **if** $Rto$, **then**
28:                   Initialize $Cwnd = 1$, $Smss = 1$
29:                   Send $Smss$
30:                **else**
31:                   Stop
32:                **end if**
33:             **end if**
34:             //Slow start phase
35:             **else** $Cwnd < Sthresh$ **then**
36:                $Cwnd = Cwnd + Smss$
37:                $Smss = Cwnd$
38:                Send $Smss$ and wait for Ack
39:             //Congestion avoidance
40:             **else** $Cwnd \geq Sthresh$ **then**
41:       $Cwnd+ = SmCss$
42:             $Smss = Cwnd$
43:             Send $Smss$ and wait for Ack
44:          **end if**
45: **end while**
```

Initially, packet priority should be set with respect to the packet type. The packet priorities are assigned based on their importance, and they are low for FTP, e-mail, and HTTP; medium for video; and high for voice packet types. Based on the proposed priority-processing approach, the packet with the highest priority will be executed first, followed by the less-priority packets (lines 2 to 10). The priority scheduling algorithm's priority function will be initiated as given in Line (1). If the priority function is ADD, all the packets will be added to the buffer based on their packet priority; if the priority function is GET, the packet will be retrieved from the buffer based on the packet priority.

The *Cwnd*, *Smss* and *Sthresh* are initialised (lines 11–12) before the slow start phase and then followed by sending maximum segment size (*Smss*). A slow start phase in the congestion control mechanism avoids sending more packets than the network's forwarding capacity. It avoids network congestion. In the slow start phase, whenever the packet receives an acknowledgement, the size of *Cwnd* increases by 1 *Smss*. The slow start phase will continue until a certain *Sthresh* is achieved (lines 35 to 38). After receiving the *Sthresh*, it will enter the congestion avoidance phase condition. During this phase, when a *Sthresh* of 64 KB is reached, the congestion window size increases linearly after getting a new Ack every time (Lines 40 to 43). If it receives three *Dack* (line 17), the congestion avoidance phase ends, further invoking a fast-retransmit phase (line 18). After receiving three *Dack*, the fast-retransmit phase assumes packet loss and invokes the fast recovery phase (line 19). The fast recovery phase (lines 20 to 33) sets the value of *Sthresh* equal to the max (*FlightSize*/2, 2* *Smss*), where *FlightSize* is the amount of outstanding data in the network and sets the *Cwnd* equals to Sthresh + (3* *Smss*).

After receiving an Ack for all packets, the full acknowledgement, *Cwnd* can be set to *Sthresh* while exiting the fast recovery. Then, the algorithm enters the slow start process again. Receiving a partial acknowledgement means another packet loss may occur, and this retransmits the next loss packet. If *Rto*, it sets the recover variable equal to the highest number of packets transmitted, exiting from the fast-recovery algorithm, invoking the slow start algorithm and set *Cwnd* = 1 *Smss* = 1. This process is iterative between slow start, congestion avoidance and fast recovery.

4.3. Introduction to Dynamic Controller Mapping in eSDN (dSDN)

The dSDN works on the dynamic selection of SDN controllers for load balancing and has been applied to a multi-controller distributed environment. Initially, in dSDN, all SDN controllers listen to their nearby controller and calculate the distance from a node to the controller, and the sensor to the controller, in a heterogeneous network. To enhance the network performance by reducing delay in the dSDN, the algorithm selects the shortest distance from the source node to the destination node. The load of the SDN controller is also considered when choosing the shortest path. The overall dynamic controller mapping procedure starts with fetching the controller id and device id, and the traversing will start whenever there is a new entry in the SDN router. The router table gets updated, stores the event activity, and the route is occupied. Thus, for every new entry, it fetches the associated controller device id and checks the payload of the related controller. If the load is high, another controller's information will be collected. This step continues until it finds the nearest controller with a load less than the threshold so the traffic can be redirected to that controller.

Figure 5 shows the dynamic controller mapping approach to balance the controller's load with the flow fluctuation, while the flow chart for the dSDN process is presented in Figure 6. Table 2 shows the algorithmic notations for dSDN.

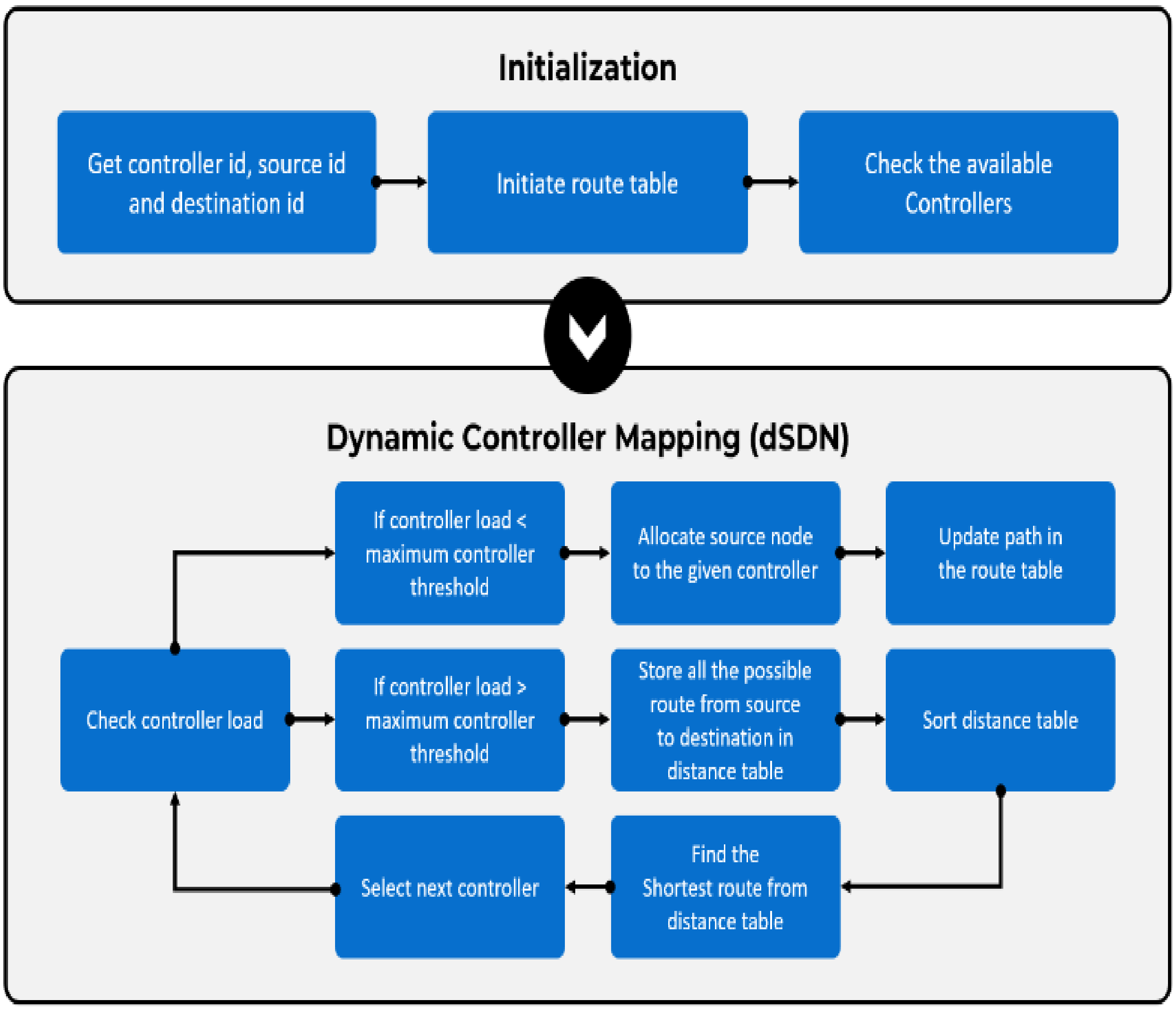

Initialization
Get controller id, source id and destination id
Initiate route table
Check the available Controllers
Dynamic Controller Mapping (dSDN)
If controller load < maximum controller threshold
Allocate source node to the given controller
Update path in the route table
Check controller load
If controller load > maximum controller threshold
Store all the possible route from source to destination in distance table
Sort distance table
Select next controller
Find the Shortest route from distance table

Figure 5. Block diagram for dSDN.

Table 2. Notations of Algorithmic Pseudocode for dSDN.

Notations	Description
βmax	Controller threshold
Cid	Controller *id*
Sid	Node source *id*
Did	Node destination *id*
Dcs	Distance from the controller to the source node
Ddc	Distance from the destination node to the controller
Rt	Routing table
Li	Load in the controller *i*
Dall	Distance table of all controllers from source to destination node
Cx	Next, selected controller.
Et	Event at time *t*
N	Total number of controller devices available in the network

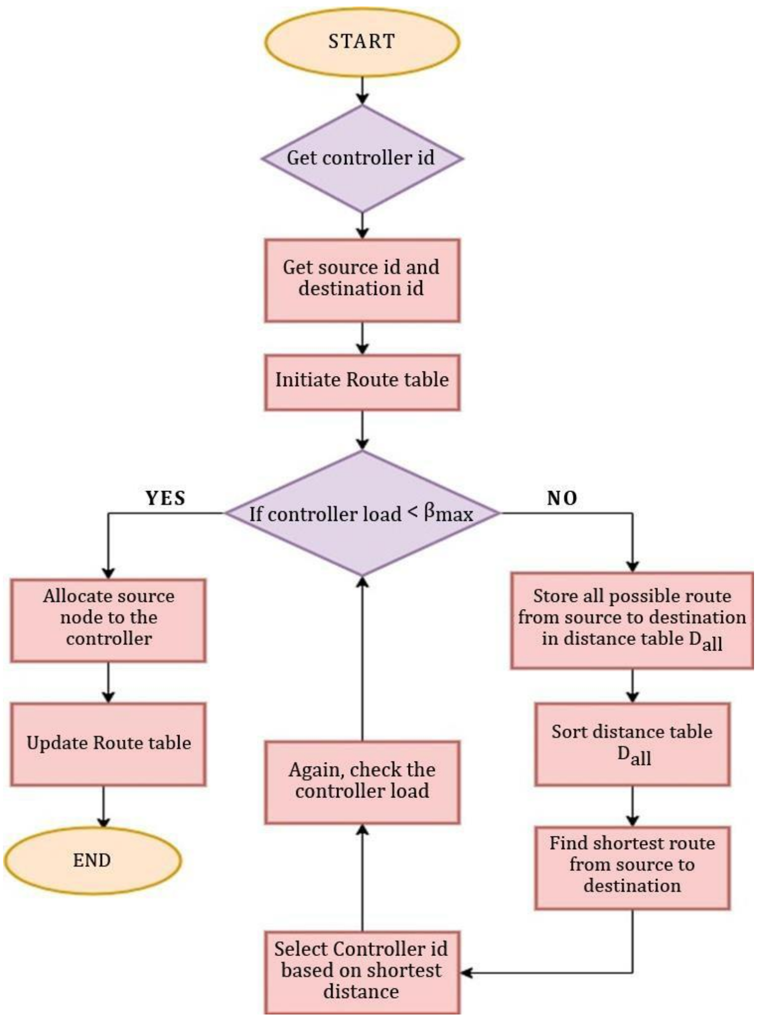

START
Get controller id
Get source id and destination id
Initiate Route table
If controller load < βmax
YES
NO
Allocate source node to the controller
Update Route table
END
Store all possible route from source to destination in distance table D_all
Sort distance table D_all
Find shortest route from source to destination
Select Controller id based on shortest distance
Again, check the controller load

Figure 6. Flowchart for dSDN.

Algorithm 2 shows the Pseudocode for the dynamic selection of SDN controllers for load balancing. The procedure starts by gathering and storing all information in the local database until propagation starts. Once communication stabilises, the information in the local database updates simultaneously along with the routing table. Critical parameters for selecting the new route include the SDN controller load, the distance of the SDN controller from the source node to the destination node, queue length at the controller, and waiting time. All this information is stored in the local database corresponding to the controller *id*, node *id* and sensor *id*.

Algorithm 2 Algorithmic Pseudocode for d SDN

```
1: if {$Cid$} then
2:     Get $Sid$, $Did$
3:     for $i$ from 1 to $N$, do
4:         if $Li < \beta max$ then
5:             $Sid$ allocates to $Ci$
6:             Update $Rt$
7:         else
8:             for j from $i + 1$ to $N$ do
9:                 $Dall = Dj, s + Dd,Cj$
10:            end for
11:            Sort ($Dall$)
12:            for $k$ from 1 to length ($Dall$) do
13:                $Cx \leftarrow Dall[k]$
14:                if $Li < \beta max$ then
15:                    $Sid$ allocates to $Cx$
16:  Update $Rt$
17:                    Break
18:                end if
19:              end for
20:            end if
21:        end for
22: end if
```

The dSDN algorithm initiates by getting the flow's controller id and corresponding source id. Lines 3 to 7 in Algorithm 2 initiate the routing table, check all the available controllers, and compare the actual controller load within its 70% threshold load. If the load on the SDN controller is less than its threshold load, then the source node is allocated to the given controller, and the path will be updated in the routing table. However, if the load on the SDN controller is greater than the threshold, a new search is initiated to find the smallest alternative route from source to destination, as seen in line 8. The distance table stores the route from source to destination in line 9. Lines 10–17 in Algorithm 2 sort the distance table to find the shortest possible route, and then the shortest route will be selected.

The dSDN is a load-aware distance algorithm, so next, it will check the controller load again before allocating the source node to the controller, as given in line 14. Supposing the controller load is less than the threshold, the source node will be allocated to the corresponding SDN controller, and the path will be updated in the routing table, as given in lines 15 and 16, respectively. Otherwise, another search will follow the same process to find the alternative path.

The approach mentioned above diverts the packets after analysing the SDN controller's load to maintain the network load so that none of the SDN controllers goes under or over-utilised, and the controller's map automatically whenever traffic flow varies. The approach also considers the distance between the source and SDN controller before allocating, so the source node must be allocated to the nearest available controller with less load than its threshold.

5. NetSim Setup

The NetSim Simulator is used as a network simulator. It allows the creation of network scenarios, traffic modelling, protocol design, and performance analysis. Many networking devices, such as switches, routers, and nodes, can be connected to design a network using network links. The various IoT components are used to establish a network. IoT is a network of various objects such as computing devices, vehicles, smart gadgets, and buildings equipped with electronic sensors or actuators using network connectivity. These objects with embedded technologies use Internet Protocol addresses for communication without human intervention. This Internet Protocol enables objects to assemble and exchange data.

Various IoT components used for IoT heterogeneous network setup are as follows [38]:

- Sensors: The sensors collect and exchange data and establish communication among the devices, leading to reliable and efficient infrastructure.
- LoWPAN gateway: It bridges the internal network with the external internet. Its role is to gather all the data from the sensors and pass the collected data to the Internet infrastructure. It has two interfaces: ZigBee connecting with sensors and WAN connecting to routers.
- Router: Connects to the cloud.
- Switch: Acts as the connecting device (used to interconnect with the server).
- Wired/wireless link: Used to establish the connection between the components.

5.1. List of Applications

The need for packet priority processing is based on the application types. Various applications are assigned different priorities based on their importance. We have mainly considered the five widely used application types: e-mail, HTTP, FTP, voice, and video. These applications and their properties are mentioned in Table 3.

Table 3. List of Application Types.

Application Type	Priorities
Email	Email applications are considered low-priority applications with an assigned priority value is 2. Email is classified under BE QoS class.
HTTP	HTTP applications are considered low-priority applications with an assigned priority value is 2. HTTP is classified under BE QoS class.
FTP	FTP applications are considered low-priority applications with an assigned priority value is 2. FTP is classified under BE QoS class.
Voice	Voice applications are considered high-priority applications with an assigned priority value is 8. Voice is classified under both rtPS and UGS QoS classes.
Video	Video applications are considered medium-priority applications with an assigned priority value is 6. Video is classified under the nrtPS QoS class.

5.2. Network Device Count

Three reference network scenarios were considered to compare and reflect on how performance degrades with the increased load on the network. The three reference networks start from a mild network consisting of a very low number of devices on the network, a moderate network comprised of the normal or standard number of devices on the network, and a heavy network consisting of a highly crowded network device. The three reference network scenarios are created by configuring the devices using various traffic generator applications. In these networks, five types of traffic were generated: File Transfer Protocol (FTP), Hypertext Transfer Protocol (HTTP), e-mail, voice, and video. Overall, two types of traffic are configured: data traffic (FTP, HTTP, and e-mail) and real-time traffic (voice and video). Table 4 shows the list of devices and applications configured.

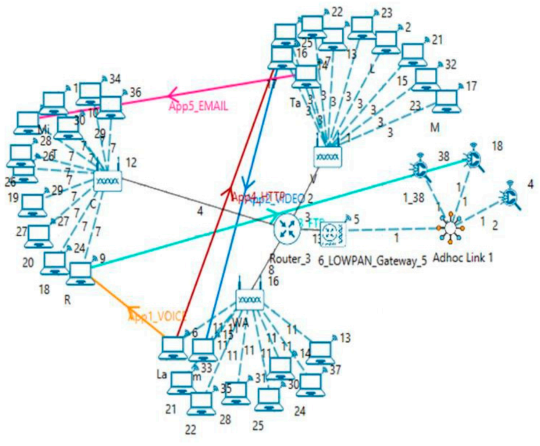
The section below describes the design and modelling of the network topologies to support the selected scenarios for the reference network.

Table 4. Device count.

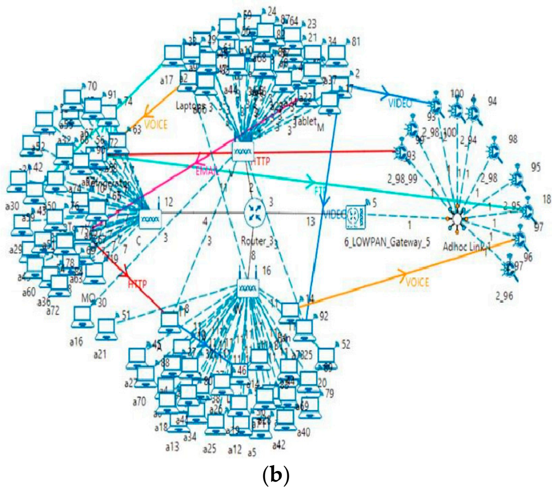
Network Type	Wireless Nodes	Applications	Wireless Sensors
Mild	30	5	3
Moderate	85	10	10
Heavy	192	20	20

5.3. Network Design and Modelling

The three network scenarios mentioned above are distinguished based on the number of connected devices, configured applications, and sensor device count. These networks work differently and provide varying results according to the generated traffic. The mild network with few connected devices can relate to previous generations when only a few smart devices were used to transfer data. The moderate network using more connected devices can be associated with the present time as many devices are connected for data transfer. The heavy network generated using many more intelligent devices represents the futuristic network with crowded IoT. The traditional networks and standard SDN in a traditional network are simulated to verify the enhancement in network performance that demonstrated better QoS. The three network topologies or scenarios for mild, moderate, and heavy traffic loads are created, configured and modelled. The network topologies for the mild, moderate, and heavy traffic scenarios are shown in Figure 7. Once the networks are modelled, the simulation is conducted to analyse the results. The following section introduces the eSDN.

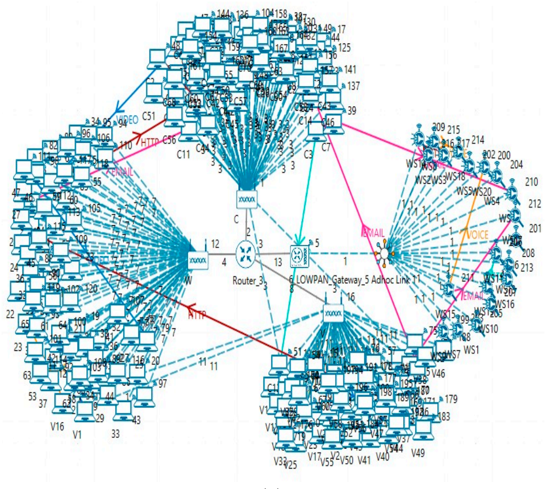


(**a**)



(**b**)

Figure 7. *Cont.*



(**c**)

Figure 7. Network topology deployment of (**a**) mild network traffic, (**b**) moderate network traffic, and (**c**) heavy network traffic.

6. Simulation of IoT Networks

The network simulation tool enables creating the network scenarios, modelling the traffic, designing the protocols, and performing analysis. It is not feasible to develop a prototype with many IoT devices; thus, this motivates us to use the simulation for our work, allowing us to capture the diversity of IoT devices to create heterogeneous traffic. In this sense, these devices are highly heterogeneous because the networks of intelligent machines exhibit different capabilities and are manufactured by diverse vendors [39]. During the simulation, the devices start communicating, and the network performance is identified in terms of delay, jitter, and throughput. It is a great challenge to provide bandwidth to all the connected devices in the existing traditional networks [40] due to the heterogeneous nature of the devices. The bandwidth usage also varies invariably when many devices are connected. We used the NetSim simulator to create and simulate the network infrastructures.

6.1. Mild Network Traffic

The mild network is the least crowded, with few connected devices. The findings demonstrate that SDN can enhance network performance by reducing delay and jitter without affecting the network throughput compared to traditional networks that can handle a small number of connected devices. However, connectivity issues arise when the number of devices grows, and it is still a challenge to maintain the network QoS due to increased real-time IoT. The eSDN enhances network performance by increasing throughput and decreasing delay and jitter. The improvement is due to processing packets per their priorities and congestion control mechanism. Thus, the network overcomes most bottlenecks and congestion issues using SDN and eSDN, enhancing the network QoS. Therefore, dSDN could not improve much throughput as the scope of improvement is minimal because the mild network is enhanced with previous contributions. However, a significant reduction in delay and jitter is noted. The results are summarised in Table 5.

Table 5. Network performance for mild network traffic.

Network Type	Throughput (Gbps)	Delay (µs)	Jitter (µs)
Traditional	0.086	98.384	16.225
SDN	0.086	71.817	14.123
eSDN	0.100	58.92	10.86
dSDN	0.100	41.45	8.79

As shown in Figure 8, using SDN in a mild network does not impact throughput; for both the traditional and standard SDN, the throughput is 0.086 Gbps. In comparison, the delay in the traditional network is 98.384 µs, which is reduced to 71.817 µs with SDN. Similarly, with SDN, the jitter is also reduced from 16.225 µs to 14.123 µs. The overall delay decreased by 27%, while the jitter was reduced by 12.95% compared to the traditional network, which implies that SDN can enhance real-time traffic. The eSDN improves the network throughput from 0.086 Gbps to 0.100 Gbps; the delay and jitter are substantially reduced compared to the SDN. In addition, there is a considerable reduction in the collided packets with the congestion control algorithm, enhancing the overall network QoS. Here, we notice that in traditional networks and SDN, the network throughput is the same, as devices are less in mild traffic, and it can be handled by the traditional network by providing the same network throughput. However, eSDN in mild network traffic significantly improves throughput. The throughput using dSDN increased by 0.16%. The delay and jitter are reduced significantly compared to eSDN. The overall delay decreased from 58.92 µs to 41.45 µs, and jitter was reduced from 10.86 µs to 8.79 µs using dSDN. The result shows that dSDN can improve network performance by removing unnecessary delays and jitter. However, the enhancement in network QoS is minimal as the scope is significantly less in a mild network.

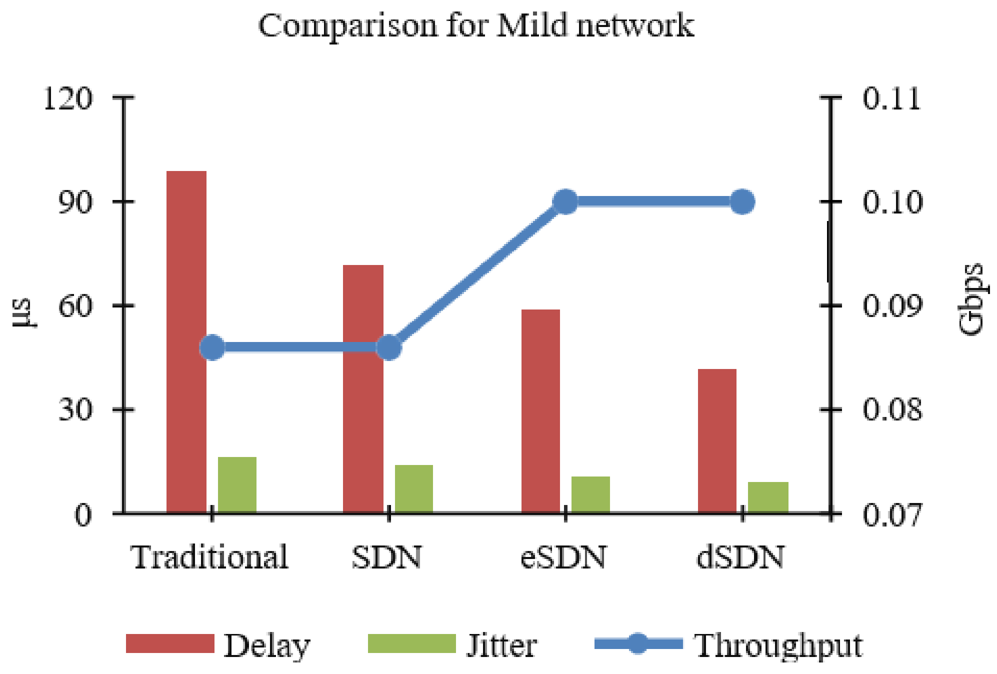


Figure 8. Simulation results for mild network traffic (overall comparison).

6.2. Moderate Network Traffic

The network with moderate traffic is more crowded than the mild one but still less crowded than the network with heavy traffic. This network has more connectivity issues, such as bandwidth starvation and QoS. It is essential to deliver packets on time to maintain the network QoS; however, due to congestion and packet loss, it becomes challenging to deliver the packets before or on time, which reduces the overall QoS of the network. Thus, eSDN improved this network's QoS by reducing network packet loss, which reduces the network delay. The moderate network has more connectivity issues than the mild, as we have analysed that the packet loss and network congestion increase as the traffic grows.

The moderate network reaches its saturation point using eSDN, and dSDN can only make minor improvements. Approximately 99.83% of network delays are already eliminated from the network by processing packets based on their priorities and using a congestion avoidance algorithm. Therefore, the eSDN is ideal for small and moderate traffic but ineffective in large-scale networks when real-time traffic varies. The results for traditional, SDN, eSDN and dSDN are summarised in Table 6.

Table 6. Network performance for moderate network traffic.

Network Type	Throughput (Gbps)	Delay (ms)	Jitter (ms)
Traditional	0.078	84.25	0.20
SDN	0.087	68.25	0.15
eSDN	0.094	0.12	0.02
dSDN	0.094	0.12	0.02

The findings illustrated in Figure 9 prove that SDN improves the network throughput and reduces delay and jitter. The throughput for the traditional network and SDN is 0.078 Gbps and 0.087 Gbps, respectively. Along with the increase in throughput, the SDN also reduces the packet delay from 84.25 µs to 68.25 µs, and the jitter is also reduced from 0.20 ms to 0.15 ms. Overall, with SDN, throughput increased by 11.66%, delay decreased by 18.99%, and jitter was reduced by 25.32% when the traffic increased; SDN enhanced QoS for real-time traffic and the data traffic.

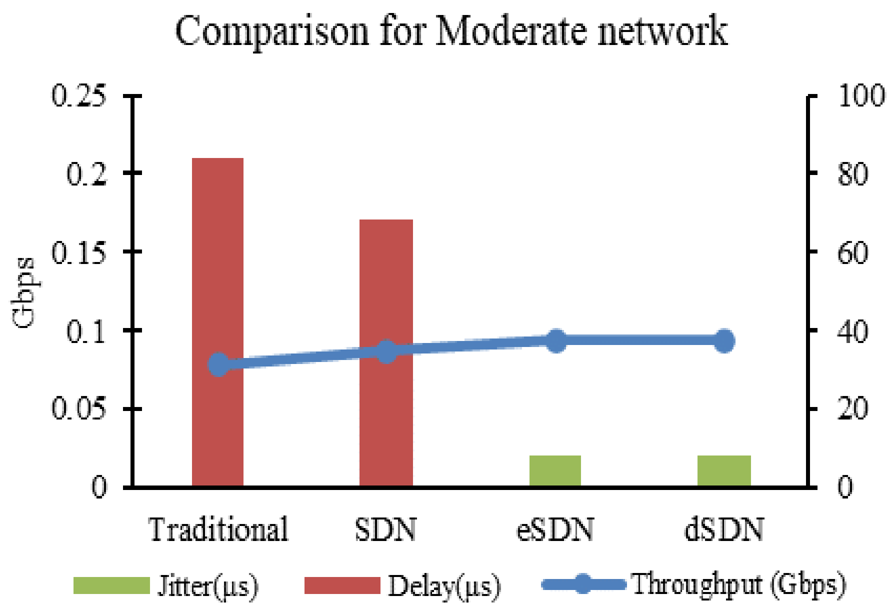


Figure 9. Simulation results for moderate network traffic (overall comparison).

The eSDN increases the network throughput from 0.087 Gbps to 0.094 Gbps, a delay is decreased from 68.25 ms to 0.12 ms, and the jitter is reduced from 0.15 ms to 0.02 ms. Thus, there is a massive reduction in delays and jitters. It is observed that eSDN most impacts this network traffic because it is a moderately crowded network, and eSDN can solve network congestion issues by lowering the packet loss with a congestion control mechanism and directing the packets as per their importance. It is subsequently enhancing the network QoS by improving the network throughput.

In dSDN, the throughput increases, and the delay and jitter are reduced. With eSDN, a significant reduction in delay and jitter has already been achieved, so there is little scope for enhancing network QoS using dSDN.

6.3. Heavy Network Traffic

The heavy network is modelled to represent the future of IoT, where the number of connected devices is increasing, and better methods are required to establish better network connectivity. For heavy traffic, the network QoS is improved with eSDN as this network is the most crowded and has larger network delays, packet loss and congestion. However, eSDN is not ideal for heavy networks due to a lack of dynamic adjusting of the SDN controller load. The dSDN enhances eSDN by adjusting the network load dynamically whenever the load fluctuates. The simulation results for traditional networks, SDN, eSDN and dSDN are shown in Table 7.

Table 7. Network performance for heavy network traffic.

Network Type	Throughput (Gbps)	Delay (ms)	Jitter (ms)
Traditional	0.048	235.32	5.56
SDN	0.048	230.95	4.31
eSDN	0.051	222.95	1.31
dSDN	0.059	36.08	0.68

The results conclude that SDN can slightly enhance the network performance by reducing the delay and jitter; however, it does not affect the network throughput, as shown in Figure 10. The throughput for the traditional network and standard SDN is the same and equals 0.048 Gbps. On the other hand, with SDN, the delay is reduced from 235.32 ms to 230.95 ms, and the jitter is also reduced from 5.56 ms to 4.31 ms. The findings indicate that SDN improves the throughput marginally by 0.27%, while the delay and jitter are notably reduced compared to the traditional network. Using SDN, the overall delay decreased by 1.86%, while jitter was reduced by 22.56% compared to the traditional network. These results demonstrated that SDN outperforms the traditional network; the best improvement in network performance is noted in the moderate network. SDN reduces mild network delay without affecting its throughput. An insignificant enhancement in network QoS is observed for heavy network traffic.

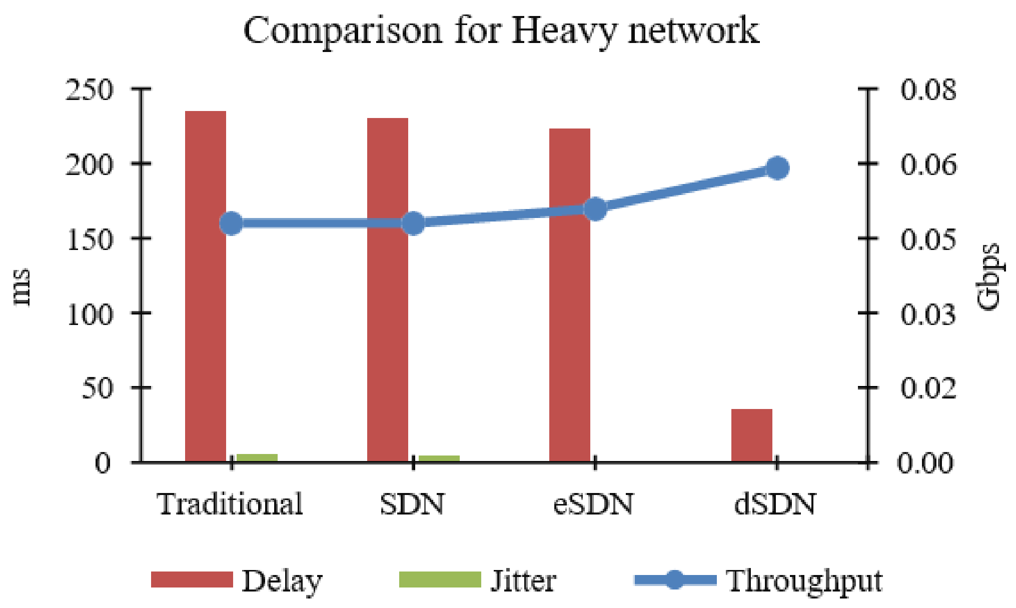


Figure 10. Simulation results for heavy network traffic (overall comparison).

The eSDN can reduce the network delay minimally compared to SDN, but only a marginal reduction in the delay is noted. The improvement with eSDN is noted in the throughput, with an increase of 7.60%, while the jitter is substantially reduced. The delay decreased by 3.46%, while the jitter was reduced by 69.70%. These findings show that although eSDN can manage the network traffic better than SDN, in large-scale networks, eSDN is less effective in improving the network QoS because eSDN is based on the static deployment of SDN controllers. Subsequently, the network controller gets unbalanced when traffic varies. Hence, we need dynamic controllers for real-time networks that must map themselves when flow varies to adjust their loads, improving network QoS.

Thus, the proposed dSDN improves the heavy network performance by increasing throughput and reducing delay and jitter. Compared to eSDN, it reduces delay from 222.95 ms to 36.08 ms using dSDN, and jitter is reduced from 1.31 ms to 0.68 ms using dSDN while improving throughput from 0.051 Gbps to 0.059 Gbps using dSDN. Therefore, the dSDN outperforms the eSDN, which is most suitable for growing IoT devices.

We further applied dSDN for 300 devices to confirm the enhancement. Our results in Table 8 conclude dSDN is the best fit for growing IoT devices. It has improved the network performance by increasing network throughput and reducing the network delay and jitter.

Table 8. Network performance for 300 devices.

Network Type	Throughput (Gbps)	Delay (ms)	Jitter (ms)
Traditional	0.38	256.35	10.89
SDN	0.39	238.89	6.85
eSDN	0.48	202.75	4.32
dSDN	0.53	45.95	1.15

7. Comparison of Results

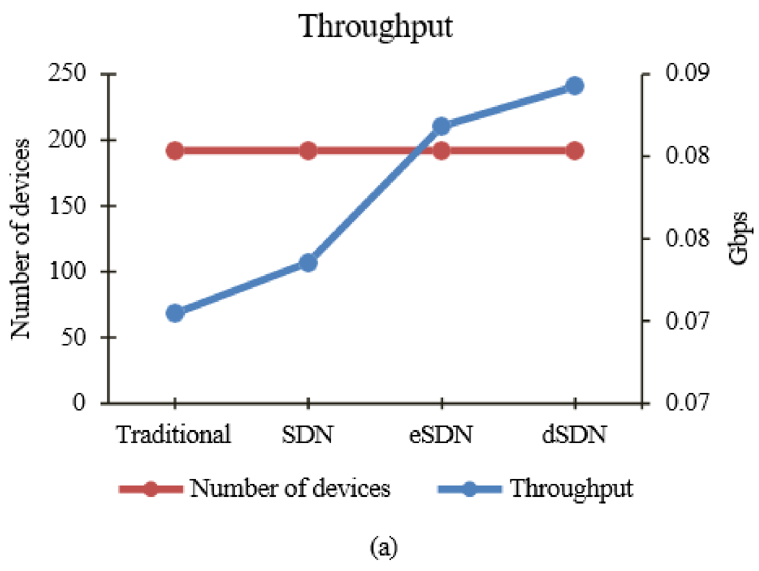
The overall network performance is compared with the number of connected devices in terms of (a) throughput, (b) delay, and (c) jitter. Note that in this section, for comparison of the results, we have taken the average value of mild, moderate, and heavy scenarios for each network performance matrix.

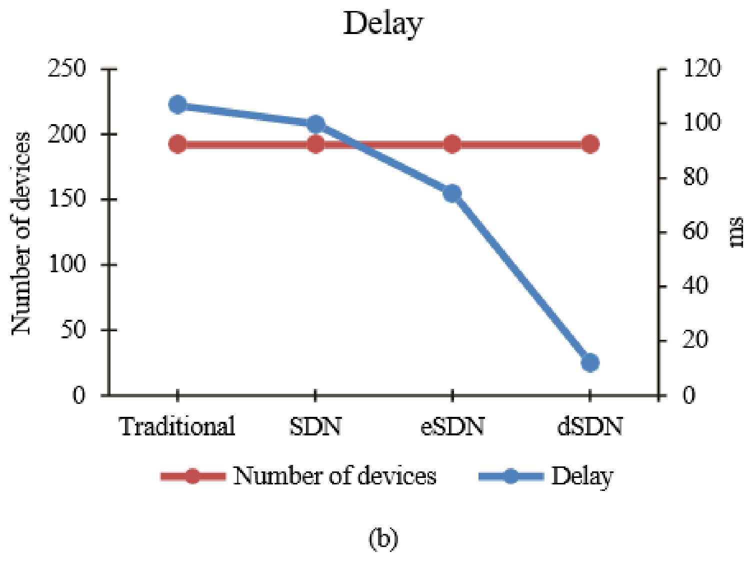
The discussion aims to observe how different approaches without SDN, with standard, eSDN and dSDN behave under similar network setups and to analyse the performance of the proposed contributions. The network is set up for 200 heterogeneous devices to analyse the system performance when the traffic is high. The proposed contributions are tested under similar network loads. Thus, we found that the traditional network without SDN is congested with bottleneck issues, and minimum throughput is noted for this network. With standard SDN, network performance is raised as it overcomes most traditional network issues such as bottleneck, mobility, and scalability. With eSDN, the network performance is enhanced further with reduced packet loss, high-priority packets reached on time, and improved overall network QoS. The dSDN supports the dynamic mapping of controllers in real-time networks and is most effective for dynamic traffic in large-scale networks.

The minimum network throughput is noted for traditional networks and the maximum for dSDN, as shown in Figure 11a. On the other hand, network QoS is progressively enhanced with the proposed approaches.

The maximum delay is noted for traditional networks, and the minimum for dSDN is shown in Figure 11b. It proves that the network congestion is reduced, and the network is scalable enough to adjust to growing connected devices.

The traditional networks have the highest jitter, which is reduced progressively using the standard SDN, eSDN and dSDN, as shown in Figure 11c. The eSDN almost eliminated the network jitter; thus, dSDN could not considerably improve it. The discussion shows that network performance gradually improves by eliminating the identified issues; for example, SDN overcomes traditional network issues. The eSDN further raised the throughput by reducing delay and jitter, showing significant improvements. Still, this was not an ideal approach for heavy traffic. The dSDN overcame the identified issues in static SDN controller deployment by proposing dynamic controller mapping. Eventually, each proposed contribution identified the limitations of previous contributions and provided the relevant solutions to overcome those limitations.





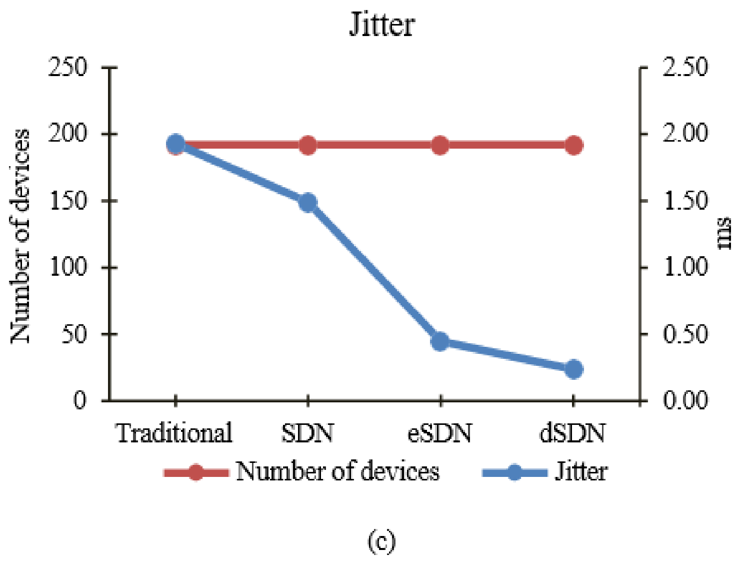


Figure 11. Network types (Traditional, SDN, eSDN and dSDN) versus connected devices: (**a**) throughput, (**b**) delay, and (**c**) jitter.

8. Conclusions

To overcome the limitations of traditional SDN in handling dynamic fluctuation in traffic, in this paper, we have introduced two models—eSDN and dSDN. The performance of these models has been tested in a simulation environment using three different types of network traffic: mild, moderate and heavy. First, simulation results for traditional and standard SDN are compared, demonstrating that standard SDN overcomes most of the identified traditional network issues. For moderate network traffic, SDN exhibits a performance enhancement in throughput by 11.66% and a reduction in delay and jitter by 18.99% and 25.32%, respectively. eSDN increases overall network QoS for all three network scenarios, but the most significant improvement is observed in moderate network traffic with a 7.88% increase in throughput, 99.83% decrease in delay, and 85.48% reduction in jitter compared to standard SDN. However, the deployment of SDN controllers in eSDN is static, which fails to accommodate the growing devices by adjusting their loads in real-time.

Consequently, it leads to a lack of load balancing among the controllers. Our proposed dSDN maintains the network load with dynamic flow fluctuation. The dynamic controller mapping in dSDN improves the overall network QoS in the heavy traffic network that emulates the futuristic network. Under heavy traffic, dSDN shows better performance with an increase in throughput by 13.63%, an 83.82% decrement in delay, and a 47.90% reduction in jitter compared to eSDN. Our future work will focus on further enhancement of dSDN by using some agent-based approach to handle the dynamic load more efficiently for the next-generation Internet.

Author Contributions: Conceptualization, A.S., V.B. and J.K.; methodology, A.S., V.B. and J.K.; software, A.S.; validation, A.S., V.B. and J.K.; formal analysis, A.S., V.B. and J.K.; investigation V.B. and J.K.; resources, A.S.; data curation, A.S.; writing—original draft preparation, A.S.; writing—review and editing, V.B. and J.K.; visualization, A.S., V.B. and J.K.; supervision V.B. and J.K.; project administration, V.B. and J.K.; funding acquisition, A.S., V.B. and J.K. 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 was generated in NetSim by configuring various networks.

Acknowledgments: The document was proofread by external services to make it grammar free.

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

References

1. Bera, S.; Misra, S.; Vasilakos, A.V. Software-defined networking for Internet of Things: A survey. *IEEE Internet Things J.* **2017**, *6*, 1994–2008. [CrossRef]
2. Benzekki, K.; El, F.A.; Elbelrhiti, E.A. Software-defined networking (SDN): A survey. *Secur. Commun. Netw.* **2016**, *18*, 5803–5833. [CrossRef]
3. Caraguay, A.L.V.; Peral, A.B.; López, L.I.B.; Villalba, L.J.G. SDN: Evolution and opportunities in the development of IoT applications. *Int. J. Distrib. Sens. Netw.* **2014**, *5*, 10.
4. Coughlin, M. *A Survey of SDN Security Research*; University of Colorado Boulder: Boulder, CO, USA, 2013.
5. Gong, Y.; Huang, W.; Wang, W.; Lei, Y. A survey on software-defined networking and its applications. *Front. Comput. Sci.* **2015**, *6*, 827–845. [CrossRef]
6. Nunes, B.A.A.; Mendonca, M.; Nguyen, X.N.; Obraczka, K.; Turletti, T. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Commun. Surv. Tutor.* **2014**, *3*, 1617–1634. [CrossRef]
7. Rowshanrad, S.; Namvarasl, S.; Abdi, V.; Hajizadeh, M.; Keshtgary, M. A survey on SDN, the future of networking. *J. Adv. Comput. Sci. Technol.* **2014**, *2*, 232. [CrossRef]
8. Scott-Hayward, S.; Natarajan, S.; Sezer, S. A survey of security in software-defined networks. *IEEE Commun. Surv. Tutor.* **2016**, *1*, 623–654. [CrossRef]
9. Yoon, C.; Park, T.; Lee, S.; Kang, H.; Shin, S.; Zhang, Z. Enabling security functions with SDN: A feasibility study. *Comput. Netw.* **2015**, *85*, 19–35. [CrossRef]

10. Kreutz, D.; Ramos, F.M.V.; Veríssimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-defined networking: A comprehensive survey. *Proc. IEEE* **2015**, *1*, 14–76. [CrossRef]
11. Karakus, M.; Durresi, A. A survey: Control plane scalability issues and approaches in software-defined networking (SDN). *Comput. Netw.* **2017**, *112*, 279–293. [CrossRef]
12. Jain, S.; Kumar, A.; Mandal, S.; Ong, J.; Poutievski, L.; Singh, A.; Vahdat, A. B4: Experience with a globally-deployed software defined WAN. In Proceedings of the ACM SIGCOMM'12, Hong Kong, China, 12–16 August 2013; pp. 3–14.
13. Cheng, T.Y.; Wang, M.; Jia, X. QoS-guaranteed controller placement in sdn. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), San Diego, CA, USA, 6–10 December 2015; pp. 1–6.
14. Dixit, A.; Hao, F.; Mukherjee, S.; Kompella, R. Towards an elastic distributed sdn controller. In Proceedings of the ACM SIGC OMM HoTSDN'13, Hong Kong, China, 12–16 August 2013; pp. 7–12.
15. Curtis, A.R.; Mogul, J.C.; Tourrilhes, J.; Yalagandula, P.; Sharma, P.; Banerjee, S. Devoflow: Scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Revolut.* **2011**, *4*, 254–265. [CrossRef]
16. Koponen, T.; Casado, M.; Gude, N.; Stribling, J.; Poutievski, L.; Zhu, M.; Ramanathan, R.; Iwata, Y.; Inoue, H.; Hama, T.; et al. Onix: A distributed control platform for large-scale production networks. In Proceedings of the OSDI'10, Vancouver, BC, Canada, 4–6 October 2010; pp. 1–6.
17. Tootoonchian, A.; Ganjali, Y. HyperFlow: A distributed control plane for OpenFlow. In Proceedings of the Internet Network Management Conference on Research on Enterprise Networking, USENIX Association, San Jose, CA, USA, 27 April 2010; pp. 1–6.
18. Fisher, W.; Suchara, M.; Rexford, J. Greening backbone networks: Reducing energy consumption by shutting off cables in bundled links. In Proceedings of the 1st ACM SIGCOMM Workshop Green Networks, New Delhi, India, 30 August 2010; pp. 29–34.
19. Mahadevan, P.; Sharma, P.; Banerjee, S.; Ranganathan, P. A power benchmarking framework for network devices. In Proceedings of the 8th International Conference Residential Networking, Aachen, Germany, 11–15 May 2009; pp. 795–808.
20. Mckeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *2*, 69–74. [CrossRef]
21. Yan, J.; Zhang, H.; Shuai, Q.; Liu, B.; Guo, X. HiQoS: An SDN-based multipath QoS solution. *China Commun.* **2015**, *5*, 123–133. [CrossRef]
22. Kim, H.; Feamster, N. Improving network management with software defined networking. *IEEE Commun. Mag.* **2013**, *2*, 114–119. [CrossRef]
23. Gude, N.; Koponen, T.; JPettit, J.; Pfa, B.; Casado, M.; McKeown, N.; Shenker, S. NOX: Towards an operating system for networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *3*, 105–110. [CrossRef]
24. Gupta, R. ABC of Internet of Things: Advancements, Benefits, Challenges, Enablers and Facilities of IoT. In Proceedings of the 2016 Symposium on Colossal Data Analysis and Networking (CDAN), Indore, India, 18–19 March 2016.
25. Kaur, S.; Singh, J.; Kumar, K.; Ghumman, N.S. Round-robin based load balancing in software defined networking. In Proceedings of the Computing for Sustainable Global Develop (INDIACom), 2015 2nd International Conference, New Delhi, India, 11–13 March 2015; pp. 2136–2139.
26. Hong, C.Y.; Kandula, S.; Mahajan, R.; Zhang, M.; Gill, V.; Nanduri, M.; Wattenhofer, R. Achieving high utilisation with software-driven WAN. *ACM SIGCOMM Comput. Commun. Rev.* **2013**, *4*, 15–26. [CrossRef]
27. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **2013**, *7*, 1645–1660. [CrossRef]
28. Oteafy, S.M.A.; Hassanein, H.S. Towards a global IoT: Resource re-utilisation in WSNs. In Proceedings of the International conference on Computing, Networking and Communications (ICNC), Maui, HI, USA, 30 January–2 February 2012; pp. 617–622.
29. Neghabi, A.A.; Jafari Navimipour, N.; Hosseinzadeh, M.; Rezaee, A. Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature. *IEEE Access* **2018**, *6*, 14159–14178. [CrossRef]
30. Isong, B.; Molose, R.R.S.; Abu-Mahfouz, A.M.; Dladlu, N. Comprehensive review of SDN controller placement strategies. *IEEE Access* **2020**, *8*, 170070–170092. [CrossRef]
31. Li, X.; Djukic, P.; Zhang, H. Zoning for hierarchical network optimisation in software defined networks. In Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; pp. 1–8.
32. 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.
33. Sanner, J.; Hadjadj-Aoufi, Y.; Ouzzif, M.; Rubino, G. Hierarchical clustering for an efficient controllers' placement in software defined networks. In Proceedings of the 2016 Global Information Infrastructure and Networking Symposium (GIIS), Porto, Portugal, 19–21 October 2016; pp. 1–7.
34. Pham, C.; Nguyen, D.T.; Tran, N.H.; Nguyen, K.K.; Cheriet, M. Dynamic Controller/Switch Mapping: A Service Oriented Assignment Approach. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 2482–2495. [CrossRef]
35. Sabitha, N.; Jayasree, H.; Parsad, A.V.K. Virtualization of Traditional Networks using SDN. *Int. J. Interdiscip. Res. Innov.* **2019**, *7*, 1–6.
36. Stiti, O.; Braham, O.; Pujolle, G. Virtual openflow-based SDN Wi-Fi access point. In Proceedings of the 2015 Global Information Infrastructure and Networking Symposium (GIIS), Guadalajara, Mexico, 28–30 October 2015.

37. Theodorou, T.; Mamatas, L. CORAL-SDN: A software-defined networking solution for the Internet of Things. In Proceedings of the 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV SDN), Berlin, Germany, 6–8 November 2017.
38. Tetcos, NetSim User Manual. Available online: https://www.tetcos.com/downloads/v12.1/NetSim_User_Manual.pdf (accessed on 25 June 2023).
39. Dai, H.; Wong, R.C.; Wang, H.; Zheng, Z.; Vasilakos, A.V. Big data analytics for large-scale wireless networks: Challenges and opportunities. *ACM Comput. Surv.* **2019**, *5*, 99:1–99:36. [CrossRef]
40. Samie, F.; Tsoutsouras, V.; Bauer, L.; Xydis, S.; Soudris, D.; Henkel, J. Computation offloading and resource allocation for low-power IoT edge devices. In Proceedings of the Paper Presented at the 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston, VA, USA, 12–14 December 2016.