BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **25**, No 2
Sofia • 2025 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2025-0015

An Analytical Study to Justify the Transformation from Traditional to Software-Defined Network in Terms of QoS Parameters

Mahmood Jalal Ahmad Alsammarraie[1], *Haeeder Munther Noman*[2], *Ahmed Hefdhi Hussein Hussein*[2], *Ali Abdulwahhab Abdulrazzaq*[2]

[1]*College of Engineering, Al-Iraqia University, Baghdad, Iraq*
[2]*Technical Instructors Training Institute, Middle Technical University, Baghdad, Iraq*
E-mails: mahmood.j.ahmad@aliraqia.edu.iq Haider_monther@mtu.edu.iq ahmed.ssal@mtu.edu.iq dr.ali.abdulwahhab@mtu.edu.iq

***Abstract*:** *Software-Defined Network is an emerging paradigm that has evolved to address weaknesses in traditional networks in recent years. The idea behind this technology is to separate the control plane from the data plane, making network management and programmability more flexible and easier. This paper aims to investigate the influence of the increasing number of pings* (*100-500*) *on two network platforms: software-defined network and traditional network. Ping is defined as a simple Internet application that lets users check whether a specific target IP address is available and able to receive requests in computer network administration. Moreover, Ping is also used as a diagnostic tool to make sure the host machine that the user is attempting to contact is up and running. The simulation was carried out using Mininet* (*the Mininet graphical user interface*) *to set up hosts and switches. Results revealed that software-defined networks improved the total number of packets received* (*22-40*) *%, average round-trip time* (*56-64) %, and reduced the total number of dropped packets* (*83-58*) *%. Therefore, it can be concluded that software software-defined network paradigm may be adopted for the network infrastructure's growing demand.*

***Keywords*:** *Software-Defined Network* (*SDN*)*, Software-defined network-based controller* (*POX*)*, Transport Layer Security* (*TLS*)*, OpenFlow protocol, Round-Trip-Time* (*RTT*)*. Packet Internet or Inter-Network Groper* (*Ping*)*.*

1. Introduction

Initiated by emerging megatrends (such as big data) in information and communication technologies, new Challenges for the Internet of the Future, like access from anywhere, high bandwidth, and dynamic management, remain crucial [1]. However, traditional approaches based on manual setup of proprietary devices are error-prone and cannot fully utilize the capacity of the physical network infrastructure [2]. Software-Defined Network (SDN) has been identified as one of the

most promising solutions for the Internet in the future. SDN has two distinct characteristics: the first is the separation of the control plane from the data plane, and the second is providing programmability for developing network applications [3]. As a result, SDN is set to provide a more efficient setup, better performance, and greater flexibility to accommodate innovative network designs. The research aims to study and clarify the concept of SDN as well as the OpenFlow protocol, which is described as an idea a radical new approach to networking and proving the benefit of moving to SDN by comparing the values of performance parameters between these networks and traditional networks [4].

In 1997, the Active Networks Group proposed an innovative approach to programmable network architecture, "Active Network", where the switches perform operations on demand. Data messages flow through it [5] to decouple network services from devices and allow new services to be loaded into the infrastructure as needed. Later, the Asynchronous Transfer Mode (ATM) was expanded to Develop Control Asynchronous transfer mode (DCAN) Networks [6]. However, the goal was to design and develop the infrastructure necessary to control and manage ATM networks. The hypothesis concentrated on the control and management functions of many devices (ATM switches in the case of DCAN) should be separated and delegated to external entities dedicated to this purpose, which is the concept of SDN. In the mid-2000s, 4D architecture was developed [7] with a clear separation between the routing decision logic and the protocols that govern the interaction with network elements, where the functions are divided. Network control is divided into four levels: decision, dissemination, discovery, and data. In 2006, the IETF NETwork CONFiguration Task Force (NETCONF) was proposed [8]. Later, in 2006, the Ethane Project [9] was proposed, which defined a new network architecture for corporate networks that consisted of a network of ethane switches that include flow tables and a control unit. Through a secure channel, the unit can control the switches' control communication and decide whether the packet should be forwarded or not. However, to move from traditional networks to SDN, it must first be proven that this transition will lead to great benefits of increasing performance, improving network behavior, and making it more resilient for testing purposes, management, etc.

2. SDN

SDN is an approach that employs software-based controllers or Application Programming Interfaces (APIs) to communicate with underlying hardware infrastructure and direct traffic on a network. However, this model differs from traditional networks, which use dedicated hardware devices (i.e., routers and switches) to control network traffic. SDN can create and control a virtual network, or control a traditional hardware – via software. While network virtualization allows organizations to segment different virtual networks within a single physical network, or to connect devices on different physical networks to create a single virtual network, SDN enables a new way of controlling the routing of data packets through a centralized server.

2.1. SDN vs traditional

The key difference between SDN and traditional networking is infrastructure; SDN is software-based, while traditional networking is hardware-based. Because the control plane is software-based, SDN is much more flexible than traditional networking as it allows administrators to control the network, change configuration settings, provision resources, and increase network capacity all from a centralized user interface, without the need for more hardware. There are also security differences between SDN and traditional networking. Thanks to greater visibility and the ability to define secure pathways, SDN offers better security in many ways. However, because SDN uses a centralized controller, securing the controller is crucial to maintaining a secure network.

2.2. SDN security threats

The logically centralized controller in SDN is responsible for making decisions regarding packet routing. Hence, it is a heavily targeted point for malicious actions and attacks within the SDN network. The main control plane's security threats are listed below:

- **Application Threats**

Management plane applications may seriously compromise SDN security. In general, the controller is in charge of application authentication, resource authorization, appropriate separation, tracking, and auditing. Therefore, before allowing access to any resource, applications must be sorted based on their security implications. As a result, the controller's northbound APIs should include a customized security check for various application types.

- **Scalability Threats**

Every new flow in the data path requires the controller to implement a flow rule; if there are too many new flows, the controller could soon experience a bottleneck.

- **DoS & DDoS Threats**

Distributed Denial of Service (DDoS) as well as Denial of Service (DoS) attacks are the most difficult network threats. The purpose behind such attacks lies in the capability to prevent authorized users from accessing network resources. DoS and DDoS attacks cannot be prevented or mitigated by using multiple controllers because this could cause all controllers to fail in sequence.

- **Strategies to Address SDN Security Challenges**

1.Adopt redundant architectural models: Use several SDN controllers to reduce the possibility of bottlenecks and failures.

2.Employ Best Practices for Security: Use encryption, access controls, and micro segmentation to safeguard the SDN environment.

3.Choose Vendor-Agnostic Solutions: Look for platforms that support open standards and provide flexibility in multi-vendor ecosystems.

- **Combining these methods** into a complete SDN adoption plan is the most efficient action. Organizations can overcome challenges and realize the full potential of SDN by coordinating workforce development, security frameworks, and technological solutions with business objectives.

2.3. How SDN works

Here are the SDN basics: In SDN (like anything virtualized), the software is decoupled from the hardware. SDN moves the control plane that determines where to send traffic to software and leaves the data plane that forwards the traffic in the hardware. This allows network administrators who use software-defined networking to program and control the entire network via a single pane of glass instead of on a device-by-device basis. There are three parts of a typical SDN architecture [10], which may be located in different physical locations:

1. Infrastructure layer. Consists of switching devices (such as switches, routers, etc.). The function of switching devices is that they are responsible for collecting network status, temporarily stored in local devices, and sending it to controllers. Network status takes account of information such as: Network topology, traffic statistics, and network usage [11].

2. Control layer. Represents the intelligence of the network. The nodes in the control layer are called controllers, which maintain an overview of the network located in the infrastructure layer. Controller nodes can also be architecturally classified into centralized and distributed. In centralized mode, one central control node sends information, such as routing and other information, to all network devices in the network; the centralized mode has the risk of a single point of failure [12]. In distributed mode, there are many control nodes connected to network devices to exchange information.

3. Application layer. Includes SDN applications usually created to satisfy user requirements [13].

SDN is based on OPF. OPF is an open standard protocol that outlines how a centralized controller governs and configures the control plane in an SDN network. The data is kept in Mac and routing tables, which are handled by a variety of complex switching and routing protocols. In conventional networks, these tables are used to create the forwarding plane. The OpenFlow protocol provides centralized, consistent rules that are capable of managing every flow table. A single or several OpenFlow switches, each with one or more flow tables, make up an OpenFlow network [14].

The OpenFlow protocol was initially developed at Stanford University in 1998 to enable researchers to run experimental protocols in campus networks. Nowadays OpenFlow protocol is added as a feature for commercial networking devices, it provides a standard, non-regulated interface for manufacturers to access switches, routers, and wireless access points. Moreover, these OpenFlow-enabled devices provide access without requiring producers to reveal the inner workings of their products.

3. Materials and methods

The work utilized-SDN-based controller of type POX. POX is an open-source controller for creating SDN applications. The actual communication protocol between the switches and controllers, OPF, can be effectively implemented with the POX. Modeling of SDN is carried out through Mininet. Mininet is a Linux-based simulation program used for rapid modeling in SDN to manage a set of hosts,

switches, routers, and links using simulation to make a single system look like an entire network. Hosts, switches, and controller modules are created using software as an alternative of hardware. For the most part, their behavior is similar to separate hardware components. Mininet provides a simple and cheap way to test networks for developing OpenFlow applications, as it allows testing of large and complex topologies without the need for a physical network to control virtualizing the network and managing it from a single console, Mininet includes a network-aware Command Line Interface (CLI), Mininet also provides a python API for creating and testing the network. Finally, the ping tool was used to help in the discovery of the status of a network device, that is, whether the device is alive or not. The research workflow goes through two scenarios:

3.1. Scenario 1. SDN topology design

The SDN topology consisted of eight OpenFlow switches S1-S8 connected to a central remote POX controller with an IP address. PORT (192.168.56.102, 6633), eight hosts: h1-h8 with IP (10.0.0.1-10.0.0.8). 100-500 pings have been generated and transmitted from h1 to h8 and repeated 10 times using the following command:

```
h1> ping 10.0.0.0.8 –l 64
```

64 bytes of data are sent in the ICMP echo request, and 66 bytes are received in the ICMP echo reply message. Simply, ping is a computer network tool used to test connectivity and the distance between two devices from each other. Technically speaking, it sends a packet of information from one device to another over a network and measures how long it takes to receive the response from the other device. This design included a loop identical to traditional as shown in Fig. 1. POX is invoked to establish the required connection within OpenFlow switches. In this scenario, POX included the following modules:

1. OpenFlow.spanning_tree. This component creates a spanning tree by using the discovery component to create a view of the network topology. It then turns off flooding on switch ports that are not part of the tree. The STP is not particularly related to this, though they aim to achieve comparable goals, which is a quite distinct approach. The Spanning Tree component responds to changes in the network topology. If a link is down, and there is an alternative link, it can maintain the connection in the network by creating a new tree that enables flooding to occur at the ports related to the alternative link. How this component builds a spanning tree is that it goes through each switch, sees who the neighbors of this switch are, and checks to see if the neighbor exists within the tree, if it does not exist, it adds the two switches (current and neighbor) to the tree in addition to ports on the connection between them. However, this component has two options that alter the start-up behaviour.

2. No-flood. This module turns off flooding on all ports as soon as a switch connects.

3. Hold-down. This module stops flood control from being changed until the full cycle of discovery has finished and all links have had a chance to be found.

4. Forwarding. l2_learning. This module enables an OpenFlow switch to function as a kind of L2 learning switch, as it installs flows that precisely match on as many fields as possible.

5. OpenFlow. Discovery. This component helps to determine network topology as it sends specifically designed LLDP messages out of OpenFlow switches [15]. When links go up or down, it raises events. Algorithm 1 outlined the SDN topology mechanism. The following command should be issued to initialize the POX controller from the controller console C0.

```
./pox.py log.level-- DEBUG Forwarding.l2_learning Openflow.spanning_tree --
no-flood --hold-down
```

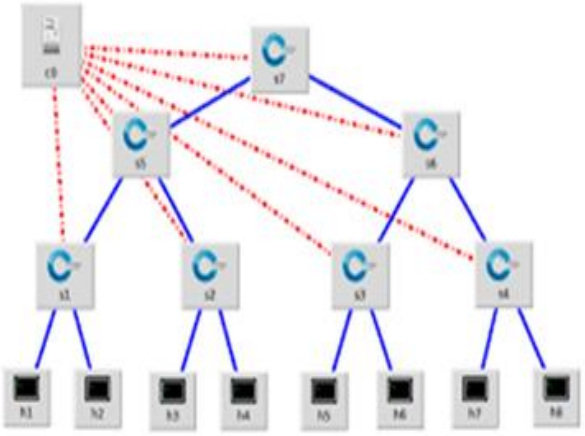

Fig. 1. SDN topology

Table 1. Algorithm 1

Algorithm 1. SDN- Topology
Input: A Flow of packets arrives at each OpenFlow switch connected with the POX controller
Output: POX controller invokes components to manage and control the network
While true do
  If there is NO match between the first packet in the flow and any flow entry in the OpenFlow switch
    Then
    Forward the packet to the POX controller.
  Else if there is a match between the first packet and any flow entry in the OpenFlow switch
    Then
    Forward the packet to the destination.
  End if
  Feed forward information back to the OpenFlow switch by POX
  Update the flow table in the switch by POX
End while
End

3.2. Scenario 2. Traditional topology design

The traditional topology consisted of eight legacy switches S1-S8 that operate as a Systematic MAC-learning switch, where the speed of each port on any switch is 10000 Mbps, eight hosts: h1-h8 with IP 10.0.0.1-10.0.0.8. Similarly, 100-500 pings have been generated and transmitted from h1 up to h8, repeated 10 times using the following command:

```
h1> ping 10.0.0.0.8 – l 64
```

The STP is not activated on legacy switches and must be activated on every switch to stop network loops while allowing redundancy. Moreover, it ensures that there is a single effective path between two switches in the network, by closing duplicate paths which may cause a loop, and when a failure occurs, alternative paths are activated by reopening the paths which has been closed. However, no controller is present in the traditional; instead, each switch operates independently. The fundamental difference between the current and previous scenarios is that the current scenario does not require a controller of any type under any platform to manage the network. Rather, it only requires a layer_2 L2 switch. Based on MAC addresses, L2 switches are excellent at quickly forwarding Ethernet communications. L2 switches build a dynamic MAC address table for effective data forwarding in the future by learning the source MAC addresses and associated switch ports from incoming data packets.

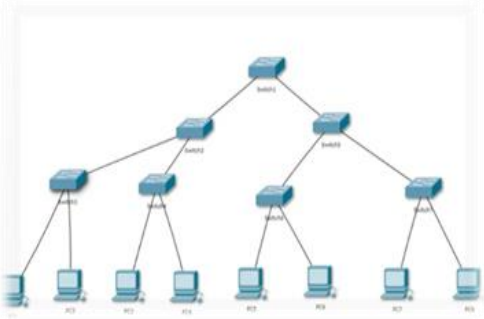

Fig. 2. Traditional topology

Table 2. Algorithm 2

Algorithm 2: Traditional Topology
While true do
  Start the blocking state on the switch port to stop forwarding frames or learning MAC addresses.
  Sleep for 20 seconds.
  Start listening state by the switch port to listen to the Bridge Protocol Data Unit.
  Sleep for 20 seconds.
  Start learning the state of the switch port to add MAC addresses to the routing table.
  Sleep for 20 seconds.
  Start forwarding state by the switch port to forward packets.
le
End

4. Experimental results

The performance of both network topologies has been evaluated and compared with emphasis on sending an increasing number of Ping (100-500) while ensuring the same spanning tree. Based on the information gathered from Table 3-5 and Figs 3-8, we may obtain the following:

- **Dropped packets**

SDN achieved better results as it recorded **5** dropped packets while traditional recorded **30** dropped packets when 100 pings were sent from h1 to h8 in both SDN and traditional. However, as the number of pings increased gradually up to 500 pings, SDN recorded an obvious reduction in dropped packets to **70** packets and **135** packets for traditional. This means that SDN develops performance as less data needs to be retransmitted

- **Average RTT**

SDN accomplished a clear improvement regarding Average RTT over traditional, as SDN started with **17.25** ms, opposite to **39.998** ms with traditional, as 100 pings are sent from h1 to h8 in both SDN and traditional. As the number of pings increased up to 500, SDN accomplished **180** ms, opposite to 500 ms in traditional **500**.

- **Total No of Pkts_received**

SDN achieved obvious progress regarding the Total number of Pkts_received. SDN recorded **90** and **600** Pkts_received, whereas traditional recorded **70** and **428** Pkts_received, as a total number of pings is 100 up to 500, respectively.

This improvement of SDN over traditional networks concerning basic network parameters is mainly due to several factors. In SDN, once the controller detects the connections between the switches then it acquires the enough knowledge of the current network status, It only works with the nearest unit and has no knowledge of the current network state, so the switches have to learn topology first, and this is what creates differences in measurements between SDN and traditional. The time switch ports take between blocking, listening, and learning states, all the way to a routing state, frame forwarding is non-existent and unnecessary with SDN topology due to the knowledge of the controller console of the current state of the network. Consequently, Fluctuations occur because in SDN, the time required to set up a connection between OpenFlow switches and the controller console to discover the connection between OpenFlow switches and build spanning trees is much less than the time required for STP to approximate switching topology in traditional networks. The results have been obtained through the percentage difference equation between SDN and traditional values to demonstrate if there is an improvement or degradation in performance, according to the following formula.

$$\text{Percentage difference (\%)} = \frac{\text{New value (SDN)-Old value (traditional)}}{\text{Old value}} \times 100\%$$

However, if the difference between the new value (SDN) and the old value (Traditional) in Tables 3-5 is positive, then there is a performance improvement. But

if the difference is negative, then there is a reduction or degradation of basic network parameters.

After doing the calculations, we discovered that the difference between SDN and traditional is positive regarding the total number of received packets and Average_RTT, which indicates that SDN improves these network parameters. On the other hand, the difference between the SDN and traditional is negative regarding the total number of dropped packets, which implies that SDN reduced or, in other words, improved this network parameter.

Table 3. Total No of Pkts_received

No of ping	Traditional	SDN	Improvement of SDN over Traditional
100	70	90	*22 %*
200	140	190	*26%*
300	270	355	*23%*
400	357	450	*20%*
500	428	600	*40%*

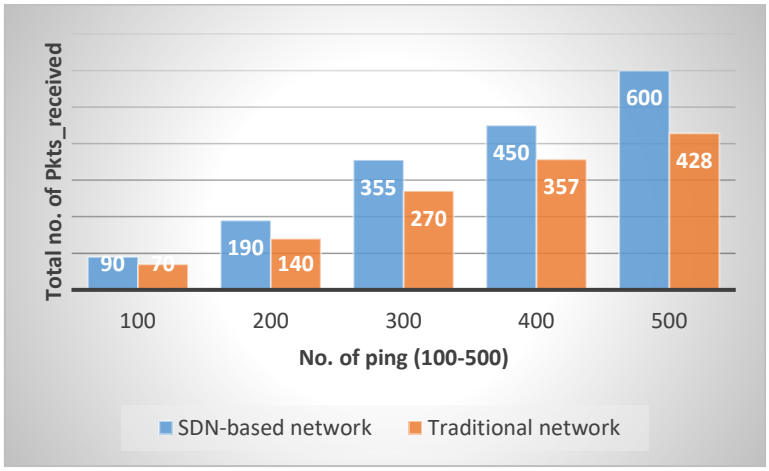

Fig. 3. Total No of Pkts_received vs No of ping

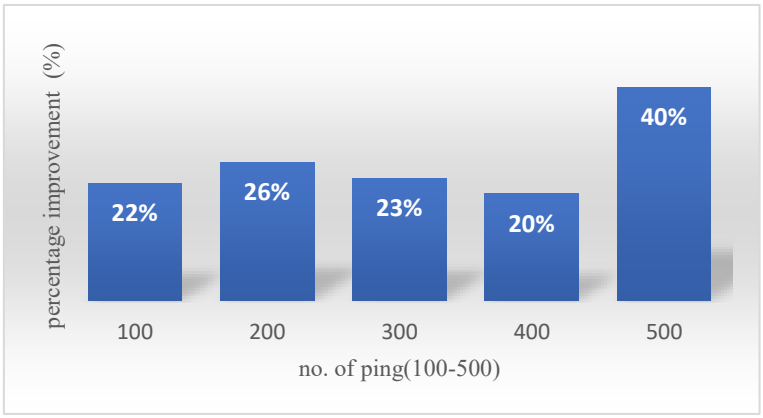

Fig. 4. Percentage improvement of SDN over traditional regarding the total No of Pkts_received

Table 4. Total No of Pkts_dropped

No of ping	Traditional	SDN	Reduction of SDN over Traditional
100	30	5	83%
200	51	9	82%
300	75	17	77%
400	125	34	73%
500	165	70	58%

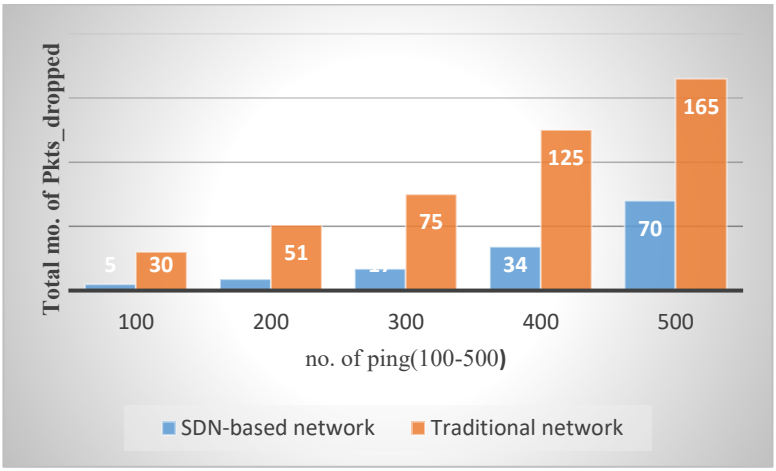

Fig. 5. Total No of Pkts_dropped vs the number of ping

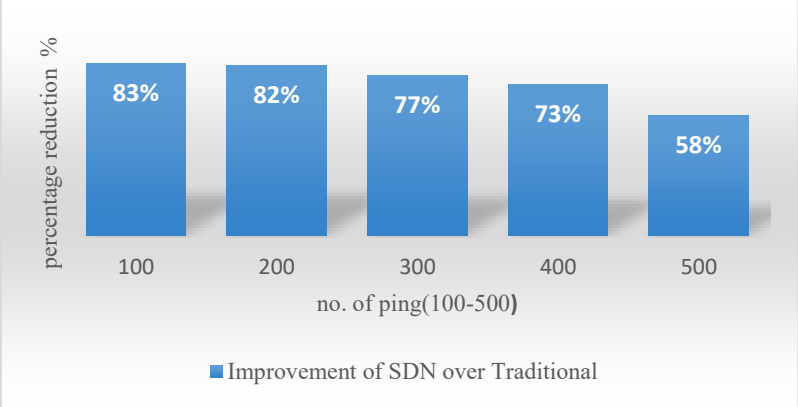

Fig. 6. Percentage reduction of SDN over Traditional regarding the Total No Pkts_dropped

Table 5. Average RTT(msec)

No of ping	Traditional	SDN	Improvement of SDN over Traditional
100	39	17	56.857%
200	80	30	62.5%
300	150	56	62.5%
400	290	100	65.517%
500	500	180	64%

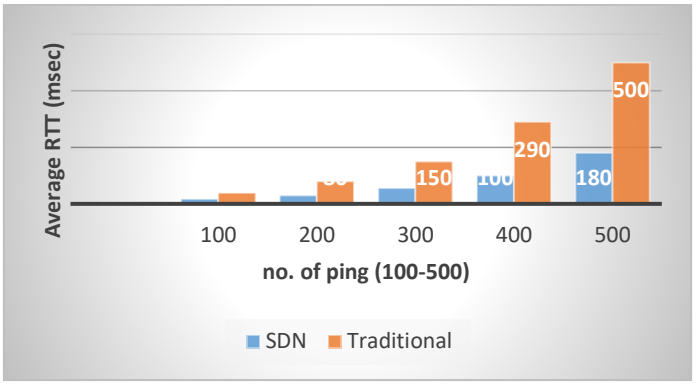

Fig. 7. Average RTT vs No of pings

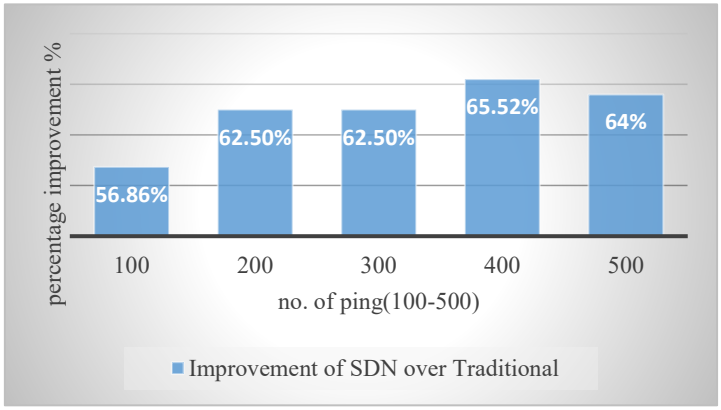

Fig. 8. Percentage improvement of SDN over Traditional regarding Average_RTT

5. Conclusion

When it comes to network performance, SDN technology has shown greater flexibility compared to traditional technology, which may be accomplished without requiring the addition of new devices or manually configuring every device, and the existence of the OpenFlow protocol enables the addition of an extra switch or router. With a centralized controller, the SDN demonstrated an improved average RTT, dropped packets, and received packets than a traditional network, keeping in mind that this study is focused on a small-tree data center network. SDN security threats such as application threat, scalability, DoS, and DDoS threats must be taken into consideration and several actions should be taken like Adopting redundant architectural models, Employing Best Practices for Security, and Choosing Vendor-Agnostic Solutions However, the most effective way to implement SDN plan that try to integrate these strategies into a comprehensive plan.

However, as a future work, it may be possible to design an SDN consisting of several controllers connected to overcome single controller limitations, while achieving synchronization between them, and then comparing performance with the traditional network.

References

1. Shamugam, V., I. Murray, J. A. Leong, A. S. Sidhu. Software Defined Networking Challenges and Future Direction: A Case Study of Implementing SDN Features on OpenStack Private Cloud. – IOP Conference Series: Materials Science and Engineering, Vol. **121**, 2016, 012003. DOI: 10.1088/1757-899x/121/1/012003.
2. Huang, T., F. R. Yu, C. Zhang, J. Liu, J. Zhang, Y. Liu. A Survey on Large-Scale Software Defined Networking (SDN) Testbeds: Approaches and Challenges. – IEEE Communications Surveys & Tutorials, Vol. **19**, 2017, No 2, pp. 891-917. DOI: 10.1109/comst.2016.2630047.
3. Jimenez, M. B., D. Fernandez, J. E. Rivadeneira, L. Bellido, A. Cardenas. A Survey of the Main Security Issues and Solutions for the SDN Architecture. – IEEE Access, Vol. **9**, 2021, pp. 122016-122038. DOI: 10.1109/access.2021.3109564.
4. McKeown, N. OpenFlow. – ACM SIGCOMM Computer Communication Review, Vol. **38**, 2008, No 2, pp. 69-74. DOI: 10.1145/1355734.1355746.
5. Anerousis, N., P. Chemouil, A. A. Lazar, N. Mihai, S. B. Weinstein. The Origin and Evolution of Open Programmable Networks and SDN. – IEEE Communications Surveys & Tutorials, Vol. **23**, 2021, No 3, pp. 1956-1971. DOI: 10.1109/comst.2021.3060582.
6. Nisar, K., E. R. Jimson, M. H. Bin Ahmad Hijazi, A. A. Ag. Ibrahim, Y. J. Park, I. Welch. A New Bandwidth Management Model Using Software-Defined Networking Security Threats. – In: Proc. of 13th IEEE International Conference on Application of Information and Communication Technologies (AICT'19), 2019. DOI: 10.1109/aict47866.2019.8981784.
7. Lai, Y.-C., A. Ali, M. M. Hassan, M. S. Hossain, Y.-D. Lin. Performance Modeling and Analysis of TCP Connections over Software Defined Networks. – In: Proc. of IEEE Global Communications Conference (GLOBECOM'17), 2017, DOI: 10.1109/glocom.2017.8254078.
8. Review for "Controller Placement Problem during SDN Deployment in the ISP/Telco Networks: A Survey". 2023. DOI: 10.1002/eng2.12801/v2/review1.
9. Casado, M., N. McKeown, S. Shenker. From Ethane to SDN and Beyond. – ACM SIGCOMM Computer Communication Review, Vol. **49**, 2019, No 5, pp. 92-95. DOI: 10.1145/3371934.3371963.
10. Ángel Barrera Pérez, M., N. Y. Serrato Losada, E. Rojas Sánchez, G. Mancilla Gaona. State of the Art in Software Defined Networking (SDN). – Visión Electrónica, Vol. **13**, 2019, No 1, pp. 178-194. DOI: 10.14483/22484728.14424.
11. Li, T., J. Chen, H. Fu. Application Scenarios Based on SDN: An Overview. – Journal of Physics: Conference Series, Vol. **1187**, 2019, No 5, 052067. DOI: 10.1088/1742 6596/1187/5/052067.
12. Blial, O., M. Ben Mamoun, R. Benaini. An Overview on SDN Architectures with Multiple Controllers. – Journal of Computer Networks and Communications, Vol. **2016**, 2016, pp. 1-8. DOI: 10.1155/2016/9396525.
13. Wazirali, R., R. Ahmad, S. Alhiyari. SDN-OpenFlow Topology Discovery: An Overview of Performance Issues. – Applied Sciences, Vol. **11**, 2021, No 15, 6999. DOI: 10.3390/app11156999.
14. Benabbou, J., K. Elbaamrani, N. Idboufker. Security in OpenFlow-Based SDN, Opportunities and Challenges. – Photonic Network Communications, Vol. **37**, 2018, No 1, pp. 1-23. DOI: 10.1007/s11107-018-0803-7.
15. Liao, L., V. C. M. Leung. LLDP-Based Link Latency Monitoring in Software-Defined Networks. – In: Proc. of 12th International Conference on Network and Service Management (CNSM'16), 2016. DOI: 10.1109/cnsm.2016.7818442.

Received: 13.04.2025, First Revision: 30.04.2025, Second Revision: 12.05.2025, Accepted: 19.05.2025