PROPOSED NEW SD-WAN ARCHITECTURE TO FACILITATE DYNAMIC LOAD BALANCING

IKIOMOYE DOUGLAS EMMANUEL

Submitted in Partial Fulfilment of the requirements for the award of the

Degree of Philosophy

School of Science, Engineering and Environment

Table of Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF PUBLICATION

Emmanuel, I. D., Linge, N., & Hill, S. (2023, March). Analysis of SD-WAN Packets using Machine Learning Algorithm. In *2023 Conference on Information Communications Technology and Society (ICTAS)* (pp. 1-6). IEEE.

Emmanuel, I. D. & Linge, N. 2023, July). Proposed New SD-WAN Architecture to Facilitate Dynamic Load Balancing. In *2023 Salford Postgraduate Annual Research Conference*. University of Salford.

ACKNOWLEDGEMENT

One of the greatest joy in my academic journey is reaching this academic level. This achievement is not just a personal milestone for me and my entire family; but for everyone who has been part of this journey. I want to acknowledge that this achievement has been made possible solely through the grace of God and His unwavering love and mercy that have sustained me throughout this academic journey. I must also express my deep appreciation for the immense support provided by my incredible spouse, Hannah Yole Ilaya. Her boundless compassion, love, and unwavering understanding have been a source of inspiration throughout this journey. Equally important in this journey are my remarkable and God-given sons, Ayebanua and Ayebaebi Ikiomoye. They have been steadfast companions, accompanying me from the inception of this academic process until its conclusion.

Additionally, a journey like this is not navigated alone. It's enriched by the invaluable guidance, unwavering support, and profound expertise of those who have paved the way before us. My sincere gratitude goes to my exceptional supervisor, Professor Nigel Linge, and Dr. Steve Hill, whose remarkable contributions and guidance have been nothing short of extraordinary throughout my years as a Postgraduate student at the University of Salford.

I extend my heartfelt appreciation and fellowship to the entire doctoral school, school of science and engineering, and the doctoral school community, with whom I have had the privilege of closely collaborating and learning. Furthermore, my family and friends, particularly Mr. Alfred Ekeuwei, Pst. Columbus Iwowari and Dr. Rowani Odum have been a pillar of unwavering support, and their contributions have significantly contributed to the success of this research endeavour. Also, I am immensely grateful for the financial support provided by the Niger Delta Development Commission.

LIST OF ABBREVIATIONS

AIMD – Additive-Increase/Multiplicative-Decrease

API – Application Programming Interface

APM - Application Performance Management

AWS – Amazon Web Services

BGP – Border Gateway Protocol

CA- Congestion Avoidance

COVN - Controllers of Virtual Networks

CNPA - Clustering-based Network Partition Algorithm

CWND – Congestion Window

CSR – Cloud Services Router

DPI – Deep Packet Inspection

DTLS – Datagram Transport Layer Security

DTM - Dynamic Traffic Management

eBPF - extended Berkeley Packet Filter

En-SDWAN - Enhanced Software-Defined Wide Area Networks

FIB - Forwarding Information Base

GNS3 – Graphical network Simulator

GUI – Graphical User Interface

HTTP – Hypertext Transfer Protocol

IDC - International Data Corporation

IETF – Internet Engineering Task Force

IoT - Internet of Things

IP – Internet Protocol

IPSec – Internet Protocol Security

ISP - Internet Service Providers

IT – Information Technology

JSON – JavaScript Object Notation

L.H.S – Left Hand Side

LSF - Linux Socket Filtering

LSO - Lifecycle Service Orchestration

LTE – Long term Evolution

MEF - Metro Ethernet Forum

MPLS - Multiprotocol Label Switching

MSS – Maximum Segment Size

NAT – Network Address Translation

NBA - Network Behaviour Analysis

NBI - North-Bound Interface

NFV - Network Function Virtualisation

NS2 – Network Simulator

OMNeT++ - Objective Modular Network Testbed in C++

OMP - Overlay Management Protocol

ONF - Open Network Foundation

ODL – Open Daylight

OPNET - Optimised Network Engineering Tools

P2P - Peer-to-Peer

PTAA - Polynomial Time Approximation Algorithm

PPDIOO - Prepare, Plan, Design, Implement, Operate, and Optimise

QoS – Quality of Service

R.H.S – Right Hand Side

RTT – Round Trip Time

RIP – Routing Information Protocol

RIB - Routing Information Base

SaaS – Software as a Service

SDN – Software Defined Networks

SD-WAN - Software-Defined Wide Area Networks

SNMP - Simple Network Management Protocol

SPI - State Packet Inspection

SS - Slow Start

TCAM - Ternary Content-Addressable Memory

TCP - Transmission Control Protocol

TLOCs - Transport Locators

TLS – Transport Layer Security

UDP – User Datagram Protocol

URL – Uniform Resource Locator

VIM - Virtual Infrastructure Management

VLAN – Virtual Local Area Network

VPN – Virtual Private Network

WAN - Wide Area Network

ABSTRACT

SD-WAN (Software Defined Wide Area Networking) is widely regarded as the future of networking as it integrates programmable features into its architecture. However, there is currently no all-encompassing solution for traffic monitoring policies in the control plane that accounts for the dynamic behaviour of applications transported in the SD-WAN architecture. In a traditional SD-WAN controller configuration, load balancing is distributed across different WANs according to predefined policies. Furthermore, existing SD-WAN controllers do not account for the effect of load balancing on end-to-end protocol performance and vice versa. To address this gap, this study proposes a new logical SD-WAN architecture to facilitate dynamic load balancing of end-to-end protocol operations in the SD-WAN architecture for improved resource utilisation. This is achieved through interaction between the Enhanced SD-WAN controller's traffic management decisions system and end-to-end protocol performance, which requires deep packet inspection of network data flows to detect signs of congestion. The solution involves developing a traffic monitor function which is responsible for analysing, classifying, and managing traffic. Additionally, an enhanced SD-WAN controller called "En-SDWAN controller" focuses on sending route updates to edge routers to change the route of traffic flows that are experiencing congestion is implemented. The performance of the developed Enhanced SD-WAN architecture was assessed in reference to the traditional SD-WAN across various scenarios, including regular traffic, link failure, and when network congestion is experienced. In the normal traffic conditions, the performance of Enhanced SD-WAN closely resembles that of traditional SD-WAN, owing to the absence of congestion within the network architecture. The average throughput performance of Enhanced SD-WAN stands at 50.07%, while traditional SD-WAN operates at 49.93%. However, during network link failures, Enhanced SD-WAN demonstrates quicker convergence and better throughput performance compared to traditional SD-WAN by 1.46%. Similarly, during instances of network congestion, Enhanced SD-WAN exhibits an average throughput of 11.21Mbps, surpassing traditional SD-WAN's average throughput of 7.02Mbps. The implementation of both the GNS3 and Python applications validates the results and design, demonstrating consistency throughout. This outcome underscores the improved flow control of packets facilitated by route updates and dynamic load sharing implemented within the Enhanced SD-WAN controller. The implementation of a new logical architecture has enhanced the performance and efficiency of traditional SD-WAN systems by adapting to the evolving nature of applications and offering better control over network traffic.

CHAPTER ONE – INTRODUCTION

This chapter of this thesis begins by discussing the background of Software-Defined Networks (SDN), Software-Defined Wide Area Networks (SD-WAN) and related concepts of network automation. It goes on to state the research question, outlining the gap in knowledge that the research aims to address and its contribution to knowledge.

1.1: Overview

The telecommunications and networking industry is undergoing rapid changes and growth, leading to reduced capital expenses and operational costs and improved user experiences (Edwards, 2023). These rapid changes in the industry have compelled the research community to focus on network automation (SDN and SD-WAN), as programmable features in networking devices hold the future for traditional networks (Anderson, 2023).

This is evidenced in the high usage of digital applications such as Video files Sharing, Video Gaming, IP on video Demand, Internet Video, Internet of Things and Zoom. Also, in the year 2023 it is predicted by Cisco that more than two-thirds of the global population will be able to access the Internet. The number of internet-connected electronic devices is expected to triple the global population, reaching an estimated 29.6 billion devices in 2023 compared to 18.4 billion in 2018 (Cisco, 2020). Furthermore, video devices and other types of applications remain highly in demand in households today, but the future holds even greater bandwidth demands as application needs continue to evolve, beyond 2023 predictions as shown in Figure 1. It is understood the current demand of bandwidth is low compared to the future demand (Cisco, 2020; Maswood et al., 2020). Hence, the need to have programmable features into the network architecture based on the increased and unpredictable nature of the use of bandwidth in the future by networking devices.

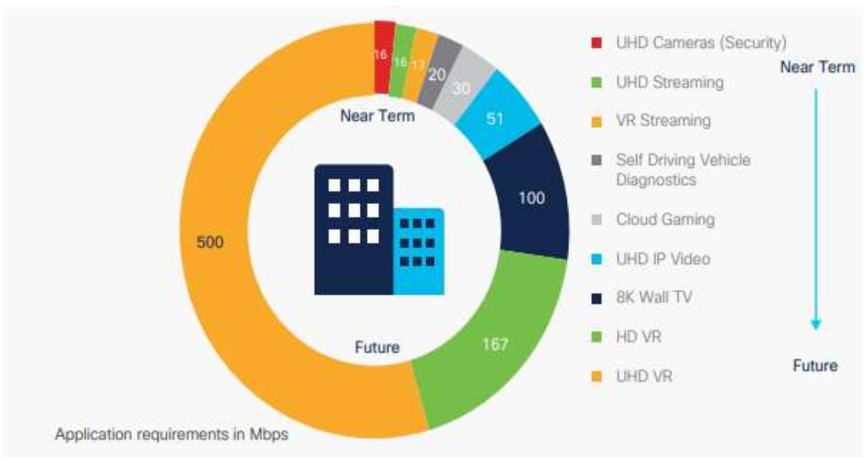


Figure 1: Applications Bandwidth requirements in the near term and future (Cisco, 2020).

The internet has become the hub of digital society where almost everything is connected from various locations, but this success comes with technical challenges such as complicated configurations, difficulty in handling network behaviour patterns, and integration issues between the control and data plane (Granath, 2023; Peterson et al., 2021). Also, traditional WANs face high bandwidth costs, limited visibility into the network, decentralised management, and lack of central control for multiple WAN connections (Ibrahim Hussein et al., 2022). Hence, traditional WANs often experience congestion due to factors like packet loss, retransmissions, poorly designed network, bandwidth failures, and video applications(Al-Saadi et al., 2019).

The growth in the development and implementation of computer networks has also come with its own set of problems (Altalebi & Ibrahim, 2022). These issues include difficulties in managing network infrastructures, cumbersome configuration arrangements that require administrators to log-in to multiple devices and high operational costs (Mazin et al., 2021; Subramanian & Voruganti, 2016). Network automation is seen as the future of network evolution, providing optimal solutions for many network issues, with a focus on efficiency in network architecture (Mohd Fuzi et al., 2021). This has led to a demand for increased agility, speed, and consistency in managing both cloud-native and traditional applications and this is where programmability and software applications are seen as a solution (Mazin et al., 2021; NetBrain, 2023).

Although the underlying technologies have evolved, network management has remained largely unchanged over the years. This is because network vendors have not focused enough on operational improvement, leading to difficulties in updating legacy and proprietary platforms which now lack automation capabilities (Lerner, 2023). According to Gartner research, around 70% of clients still use manual command line interfaces for network configuration (Lerner, 2018). As a result, it is suggested large digital businesses and infrastructures need to transform their data centre networks to be more agile and incorporate network innovation and automation mechanisms (Bar-Lev, 2022). Computer network researchers and industry experts have a role to play in this shift towards meeting the requirements for reliability, resilience, quality of service, and availability (Wikström et al., 2020).

Traditionally, computer networks are designed with the function being governed by hardware integrated into the infrastructure, making it difficult to dynamically respond to patterns of usage or new customer requirements (Bigelow, 2023). In such network designs, the data being forwarded and the control functions that govern the network operations are embedded in the same physical hardware. Such architectures lack the ability to be effortlessly reprogrammed or reassigned due to the interconnection and integration of the control and data planes within the same hardware as shown in Figure 2 (Hu et al., 2014; Maswood et al., 2020).

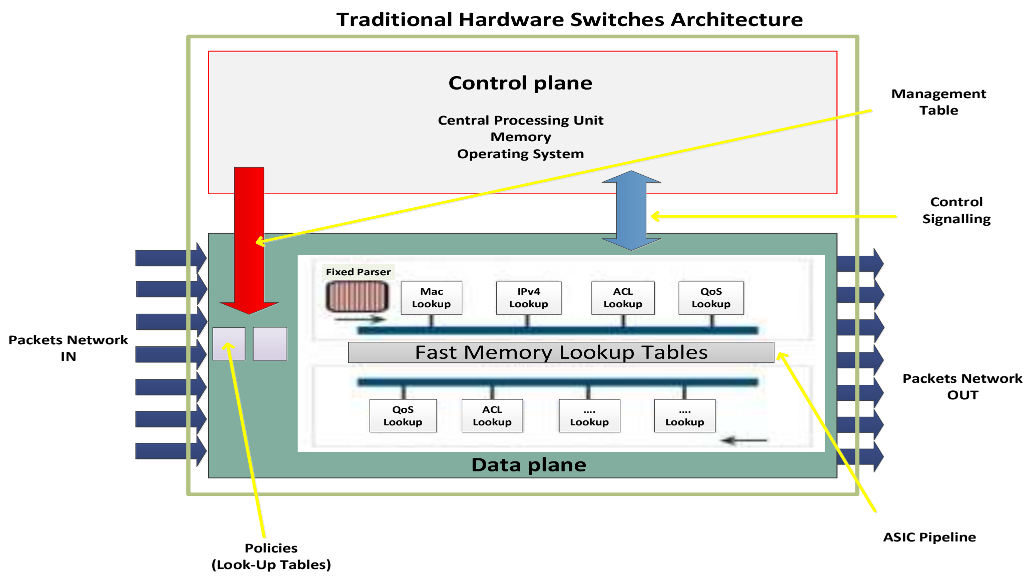


Figure 2: Traditional hardware Switch Architecture

The embedded architecture poses challenges in terms of managing the network infrastructure to dynamically adjust to varying traffic demands, complex requirements for administrators to log into various devices for configuration purposes and high operational cost (Mazin et al., 2021; Subramanian & Voruganti, 2016). Also, is the issue of network congestion caused by unstable and unpredictable traffic leading to service disruptions and latency in the network. Similarly, expanding bandwidth or adding new channels may be expensive and not a cost-effective solution (Sllame & Aljafari, 2015). A more practical approach is to incorporate programmable elements into the design, where the control plane manages packet forwarding and control, and the data plane handles data traffic.

Software-Defined Networks (SDN) is a technology that offers a new approach which aims to solve the limitations of traditional networking infrastructure. Also, make it suitable for today's business applications by separating the control plane functions (control logic) from the underlying hardware that forwards the data plane traffic hardware (Ali et al., 2020; Mohammadi et al., 2022). The separation of the data plane and control plane in SDN improves the overall network architecture by simplifying it and allowing for dynamic changes to be made easily. The data plane, which is handled by switches and routers, is responsible for forwarding traffic, while the control plane, implemented through a centralised logical controller, manages configuration, policy re-evaluation, and enforcement (Ghodsi et al., 2011; Pashkov, 2022). This separation is made possible by a programmable interface between the SDN controller and network hardware, allowing the controller to manage and secure the state of the forwarding elements in the data plane(Aditya, 2020; Koponen et al., 2010).

SDN as an umbrella concept is incorporated with other emerging technologies such as Software-Defined Wide Area Network (SD-WAN), Network Function Virtualization (NFV), and Virtual Infrastructure Management (VIM) (Landsberger, 2023). SD-WAN, which is a subset of SDN, is a promising network architecture for the future of WAN and offers a distinct approach to designing and operating network models. It utilises software-defined techniques to connect wide geographical areas and can be centrally managed and controlled through software applications (Shah, 2023; Yalda et al., 2022). SD-WAN is a cloud-scalable and zero-touch provisioning solution that is application-aware, allowing network engineers to establish wide area network connections across dispersed geographical locations through resources known as edge routers (Rohyans et al., 2019). It manages multiple connections, guarantees, and supports business-critical applications by automatically identifying the user, type, source and destination of packets, in order to minimise latency and ensure sufficient bandwidth for critical applications (Shah, 2023).

SD-WAN can be characterised by five key features: support for Virtual Private Networks (VPN) with Wide Area Network Optimisation and firewall controllers; the ability to connect to MPLS (Multiprotocol Label Switching), cellular networks and broadband/fibre connectivity, easy network management and configuration of various interfaces, and dynamic path selection, load sharing capability and resiliency in the overlay/tunnel network (Creta, 2018). The SD-WAN technology offers a reliable, fast, and cost-effective connection between branch offices and headquarters in different locations across a large geographical area. It provides optimal results for poor quality, data loss, and the impact of jitter or excess delay in critical applications, as noted by (Cooney, 2023a)

In SD-WAN architecture, load balancing can be used to prioritise critical applications by distributing network traffic across multiple paths for improved efficiency and reduced latency. This technique is commonly used in data centres and cloud applications and can help ensure an even distribution of traffic for optimal network performance. Figure 3, shows how load balancing, managed by a SD-WAN controller, can be achieved by distributing traffic between two Internet Service Providers (ISPA and ISPB). The implementation of the load balancing technique improves the overall resilience of the end-to-end communication and prioritises business-critical applications.

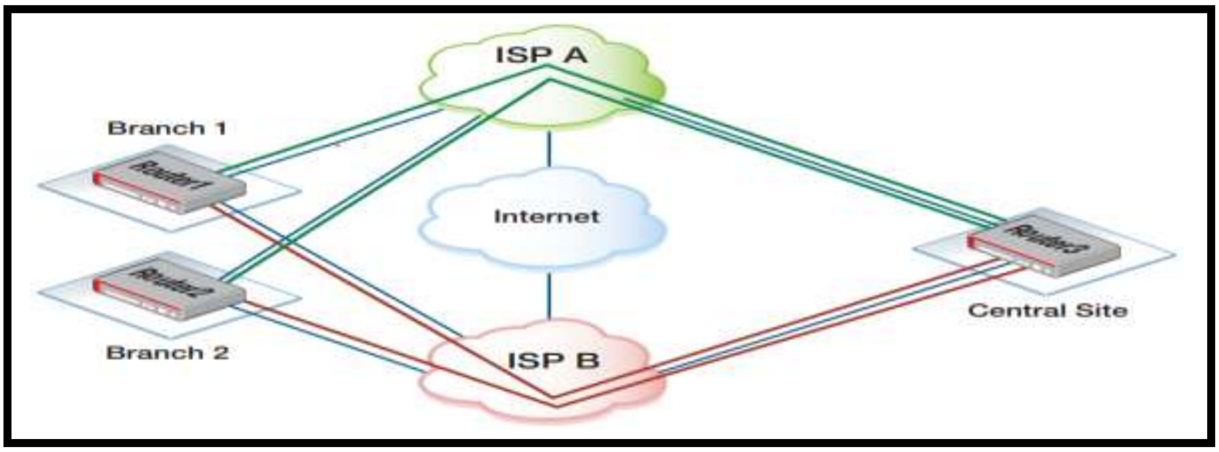


Figure 3: Graphical Illustration of Load Balancing for Business-Critical Applications

SD-WAN is widely regarded as the future of networking as it integrates programmable features into its architecture (Xie et al., 2018). However, there is currently no all-encompassing solution for traffic monitoring policies in the control plane that accounts for the dynamic behaviour of applications in the SD-WAN architecture (Joy, 2023; Versa-Network, 2023b). The complexity of monitoring is based on SD-WAN avoidance of private connections and mostly relying on connectivity from telecommunications operators which can withhold internal network traffic details. These processes make the network edge routers to act as an ambiguity leading to estimation of network traffic in an external viewpoint (Navarro et al., 2023; Raghunathan, 2021). Hence, efficient traffic monitoring is vital for the current SD-WAN architecture allowing for real-time adjustments in network performance based on the current conditions.

In a typical SD-WAN controller configuration, load balancing is distributed across different WANs according to predefined policies and a commonly used method for managing network traffic in SD-WAN architecture is the adaptive shaping (Joy, 2023; Versa-Network, 2023b). Also, this separation of the control and data planes can lead to delays in flow setup or latency, affecting throughput, and making it challenging to scale as the number of active devices increases (Bannour et al., 2017; Jammal et al., 2014; Karakus & Durresi, 2017).

Therefore, there is a requirement for an intelligent traffic monitoring system and an SD-WAN controller that can adapt to the changing behaviour of applications and increased flows from more active devices or nodes. This Thesis aims to address this requirement by conducting dynamic analysis of end-to-end protocol operations in the SD-WAN architecture to facilitate dynamic load sharing amongst various WANs. To achieve this optimal coordination between the SD-WAN controller, traffic management decisions and end-to-end protocol performance, there is a need for modification of the current SD-WAN system architecture which is the core objective of this research.

1.2: Aim and Objectives of the Study

The aim of this research is to develop a new SD-WAN architecture that incorporates a traffic monitor function that will facilitates dynamic load balancing between end-to-end protocol operation. The traffic monitor is tasked with the function of capturing and managing the traffic and sending routes updates to the controller for facilitating the load balancing amongst the various WANs.

Specifically, the research aim will be achieved by implementing the following objectives:

1. Understanding, reviewing, and presenting the state-of-the-art of previous research about SD-WAN architecture and its working operations.
2. Establishing a suitable network simulation environment using Graphical Network Simulator to design and implement the proposed new SD-WAN architecture.
3. Developing an SD-WAN controller and a traffic monitoring system that can capture and make intelligent decisions based on traffic management policies.
4. Creating and testing series of experiments using the new SD-WAN architecture in terms of network throughput and overall functionality plus check for validation of the network design.
5. Collect measurable and empirical evidence of results from the series of testing framework in the network emulator of the new SD-WAN architecture.
6. Examining the network performance from evaluating the end-to-end protocol performance that align with enhancements in throughput and reduction in congestion.

1.3: Research Question

To address the recognised gap in the current SD-WAN design and operation outlined in the previous section, the following research question will be addressed:

<div align="center">

Can the performance of SD-WAN load balancing be improved by dynamically analysing the operation of the end-to-end protocol?

</div>

In answering this question there is a need to study the relationship between traffic management decisions in SD-WAN and the performance of end-to-end protocols. That in turn will require performance metrics to be examined within the end-to-end applications including throughput, latency, jitter, and congestion. Traffic analysis also implies a need for Deep Packet Inspection (DPI) which raises a supplementary research question which is what scale of packet inspection is required to improve the load balancing of an SD-WAN.

1.4: Research Contribution

This thesis proposes a new logical architecture for SD-WAN in which a traffic monitoring process (classification, management, and analysis) is introduced between the SD-WAN edge routers and controllers. The traffic monitor provides a deep packet inspection on end-to-end application traffic and provides the SD-WAN controller with status updates which forms a new input to its load balancing algorithm in real time. These updates allow the SD-WAN to respond dynamically to the demands of applications and deliver an improved quality of service for the applications. Hence, this research contributes to the associated body of knowledge in the following ways:

1. The research presents a new logical architecture for the current SD-WANs architecture in which a Traffic Monitor function is introduced between the SD-WAN edge routers and SD-WAN controller to enable it to respond dynamically to the real-time requirements of end-to-end applications.
2. The functions of the SD-WAN traffic monitor are defined in terms of its requirements for deep packet inspection of end-to-end traffic flows and the SD-WAN controller is defined in terms of the information that must be exchanged.
3. To facilitate dynamic load balancing for the underlay networks based on the information exchange between the traffic monitoring system and the SD-WAN controller, which is based on route update obtained from the traffic monitor.

1.5: Research Methodology

The present study adheres to a structured scientific methodology depicted in Figure 4. The methodology involves posing a research question, conducting background research, performing experiments, analysing the collected data, and drawing conclusions to align or refute the hypothesis. Subsequently, the findings are communicated as result (Wright, 2023). The research process starts by identifying the research problem, followed by an extensive literature review in the domain of SD-WAN and related technologies. This review aids in formulating research questions that guide the entire study. Also, an extensive testing and network simulation process is undertaken to implement the formulated research question. The research question focused on integrating a Traffic Monitor function into SD-WAN architectures to effectively manage and control traffic flows through Deep Packet Inspection and information exchange within the SD-WAN controller.

Once the procedure is operational, data collection and analysis are conducted to test the accepted research question and supported by the results, they are communicated. However, if the research question is not supported, it necessitates the rejection of the research question, leading to a re-evaluation of the experimental or testing process.

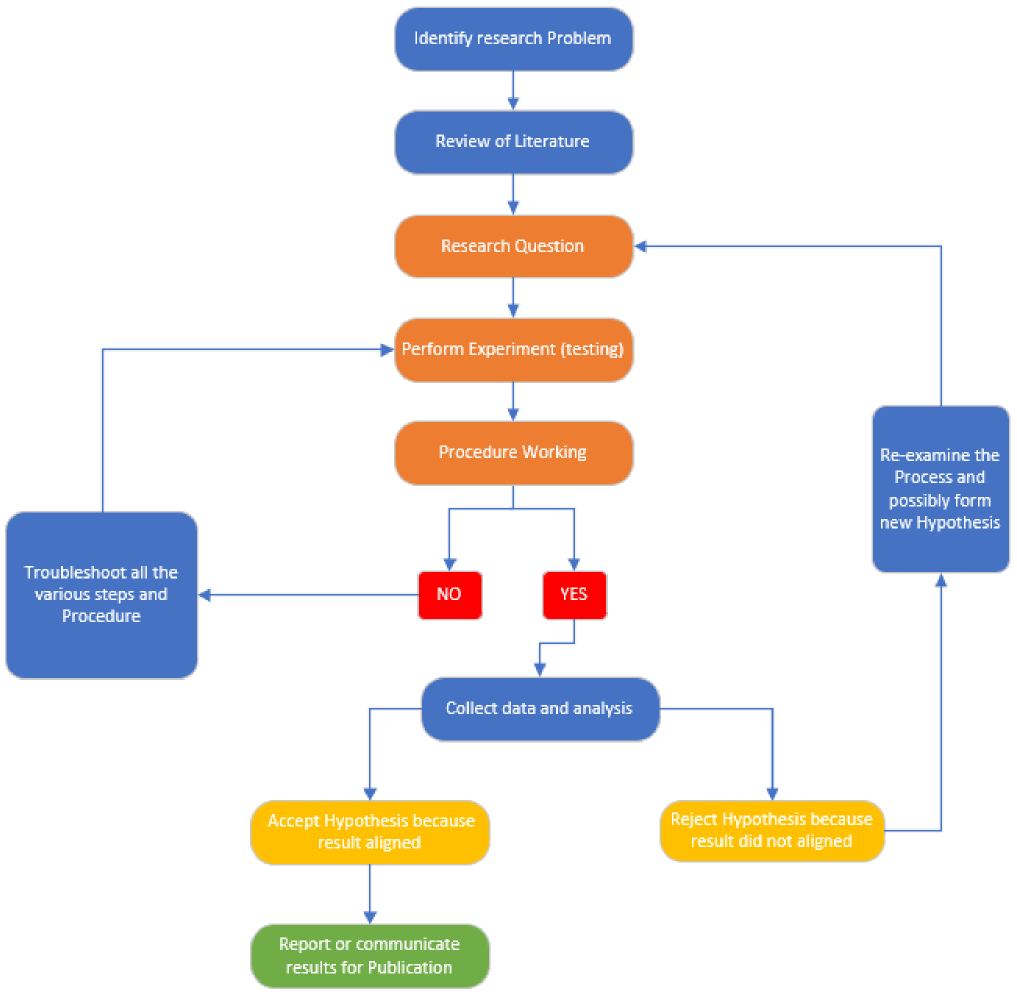

Identify research Problem
Review of Literature
Research Question
Perform Experiment (testing)
Procedure Working
Re-examine the Process and possibly form new Hypothesis
Troubleshoot all the various steps and Procedure
NO
YES
Collect data and analysis
Accept Hypothesis because result aligned
Reject Hypothesis because result did not aligned
Report or communicate results for Publication

Figure 4: Research Methodology for the Proposed Research

1.6: Thesis Outline:

This Thesis is structured to have Chapter One to Chapter Seven and each chapter is written to connect with the previous chapter to ensure the flow of conversation. This chapter is termed an introduction chapter which describes the concept of the research (SDN and SD-WAN), identifying the research gap and the contribution to the body of knowledge. **Chapter two** is described as the background and literature review about SDN and SD-WAN. In this chapter previous work related to the subject area, historical background, and key industrial perspective of the body of knowledge on the area of SDN and SD-WAN is presented. **Chapter three** focuses on the industrial perspective and the related academic work which provide the foundation for the next chapter. **Chapter four** presents the system description of the network architecture and review of the congestion control algorithms. The system design is meant to be implemented using GNS3 which is a network emulator used in the design process of the SD-WAN architecture plus setting the simulation parameters. **Chapter five** is the implementation of the system design of the research work, description of the python programming files and the various components selection that is used in the implementation of the research work. Furthermore, the programming flowchart, structure, and mathematical model implementation of the entire research work. **Chapter six** presents the results obtained in the implementation of the proposed architecture. The various results are evaluated and discussed meeting with the aim and objective of the research. **Chapter seven** is the conclusion and recommendation of the research work. In this section**,** a summary of the entire research work, clearly presenting the contribution of knowledge in the research work and recommending further work required in the research.

1.7: Summary

This chapter introduced the concepts of SDN, SD-WAN and outlined the purpose and research question of the research. A scientific methodology was discussed as the approach for conducting the study. This sets the stage for a comprehensive examination of the literature on SDN and SD-WAN in the next chapter.

CHAPTER TWO: BACKGROUND OF SDN AND SD-WAN

Chapter two of this thesis focuses on the background of Traditional WAN, SDN, SD-WAN and is divided into two sections. The first section provides an overview of the background of traditional networking, SDN and SD-WAN, while the second section covers SD-WAN load balancing, and the standardisation of SD-WAN.

2.1: Overview

The need for a software-defined approach or network function virtualisation, has become increasingly crucial in various industries such as engineering, telecommunication, computer networking, medicine, finance, and the economy, delivering numerous solutions and optimal results (Papavassiliou, 2020). SDN was developed in recent years and has evolved since 1996, allowing for effective management of forwarding nodes by network administrator's (Sezer et al., 2013). The core concept behind software-defined networking is the separation of the data and control plane in the network architecture. This separation is achieved through different phases of SDN implementation, each with its own centralised control algorithms, resulting in the creation of the SDN protocol (Gkioulos et al., 2018).

2.2: Traditional Networking

Conventional networking is designed to function on physical infrastructure that includes switches and routers. As a result, network administrators need to interact with the network infrastructure manually to configure and establish policies according to different network conditions (Keary, 2020). This entire procedure is not only time-consuming but also resource-intensive and is prone to errors that can result in higher operational expenses and numerous hurdles to overcome. Additionally, networking devices such as switches and routers were traditionally constrained and controlled by various manufacturers, making it difficult to program and re-program in meeting diverse network needs (Liang & Li, 2018).

In addition, conventional networks have a restricted ability to adjust to the high demands of Internet of Things (IoT), multimedia and other real-time applications. This is due to the vast number of devices that require connectivity and the need for instantaneous transmission of information between numerous servers. Also, the conventional architecture may not efficiently handle the scalability needs, smooth network traffic, and diverse routing requirements(Bing, 2008; Farhady et al., 2015). This has been one of the reasons for the issue of network congestion over the years.

The growing challenges faced by traditional architecture and the high demand for IoT have led to a need for innovation and transformation in IT infrastructure. Researchers have identified that the traditional networking system may not be able to meet the increasing demands and requirements needed to scale up consistently with the evolving and demanding operations of IoT, cloud computing, and artificial intelligence. As a result, there is a pressing need to adapt to these changes and accommodate the IoT requirements, multimedia, and other real-time applications (Feamster et al., 2014; Huang et al., 2010). For example, consider packet forwarding, where data packets are delivered to their destination based on pre-programmed rules and routing policies in traditional networking architecture.

While some devices can perform traditional routing methods like dynamic and static routing, others can carry out more complex configurations like QoS Classification and Scheduling. However, these limitations can negatively impact stability, efficiency, security, scalability, and dependability when there is an increase in traffic volume (Hu et al., 2014). In today's networking landscape, it's challenging to modify the traditional switch or router design due to the three separate planes - data plane, control plane, and management plane - as shown in figure 5. This division of the planes can make it difficult to reconfigure or re-task the traditional design.

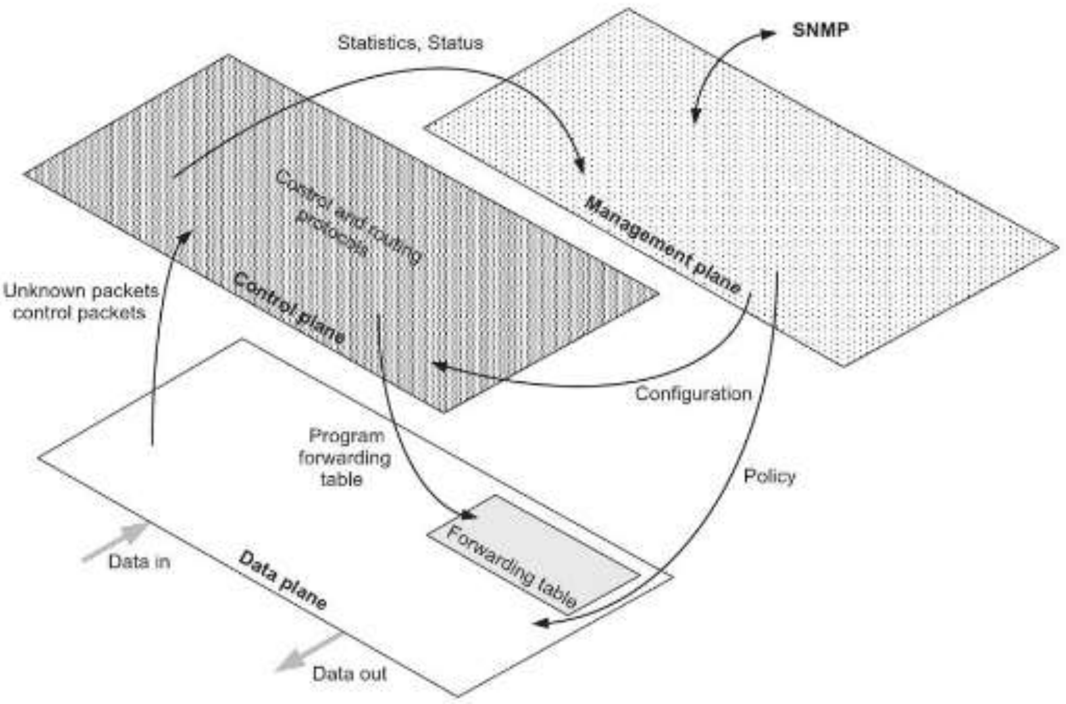


Figure 5: Traditional switch architecture (Goransson et al., 2016)

The data plane is responsible for receiving and transmitting packets using the forwarding table. It also manages tasks such as packet scheduling, buffering, header modification, and packet forwarding. If packet information is not captured in the forwarding table, it will be modified and added to the existing table. Meanwhile, the control plane maintains an updated record of the forwarding table and processes various routing protocol requests in a timely manner. The configuring, overseeing, managing, and monitoring of the services by network administrators in each of the layers defines the management plane. The SNMP (Simple Network Management Protocol) is an active tool and standard protocol that can be used in the management and monitoring of networking components at this layer (Al-Haddad & Velazquez, 2019; Goransson et al., 2016). To tackle these challenges, it's essential to shift towards software modules instead of rigid, hardware-embedded systems. This is where SDN comes in, and it is believed that this approach can substantially enhance network architecture and performance by enabling better access or flexibility and resource allocation. It can also improve the quality of service, security, and dependability of the overall network architecture.

2.3: Software Defined Networking

In 2009, a solution using OpenFlow at Stanford University gave birth to Software-Defined Networking. This innovative approach highlighted the separation of the data plane and network control plane, with the latter in charge of managing and directing devices. Major companies such as Cisco, Silver Peak, Fortinet, Aryaka, and Riverbed have adopted SDN. For instance, Nokia is leveraging the technology to virtualise the access network function and move its broadband network to the cloud, thereby revolutionising the telecom industry (Fruhlinger, 2023; Kannan, 2023). The SDN framework is known for its cost-effectiveness, flexibility, and adaptability, making it a solution for high-bandwidth utilisation in today's network architecture (Eissa et al., 2019). The four key concepts that highlight's Software-Defined Networking are:

- ➢ SDN involves separating the data and control plane, with the control plane functions being completely removed from the underlying infrastructure and the data plane focused solely on forwarding traffic (Kreutz et al., 2014).
- ➢ The control plane forwarding decisions are based on flows, which are sequences of packets between a source and destination, instead of destinations, with all flows configured to have the same service policies (Jamjoom et al., 2014).
- ➢ The control logic is moved to an abstract layer called a network operating system or SDN controller, which is a software-based model that provides essential abstractions and resources for programming forwarding devices, operating on server technology (Kreutz et al., 2014).
- ➢ The SDN controller can be re-programmed through a software application platform and easily interacts with the underlying infrastructure (Kreutz et al., 2014).

Understanding the history and principles of SDN is crucial for shifting from conventional networking to a software-based approach suitable for modern networking applications. Figure 6: depicts this transition from traditional networking to a software-defined network architecture in a visual format.

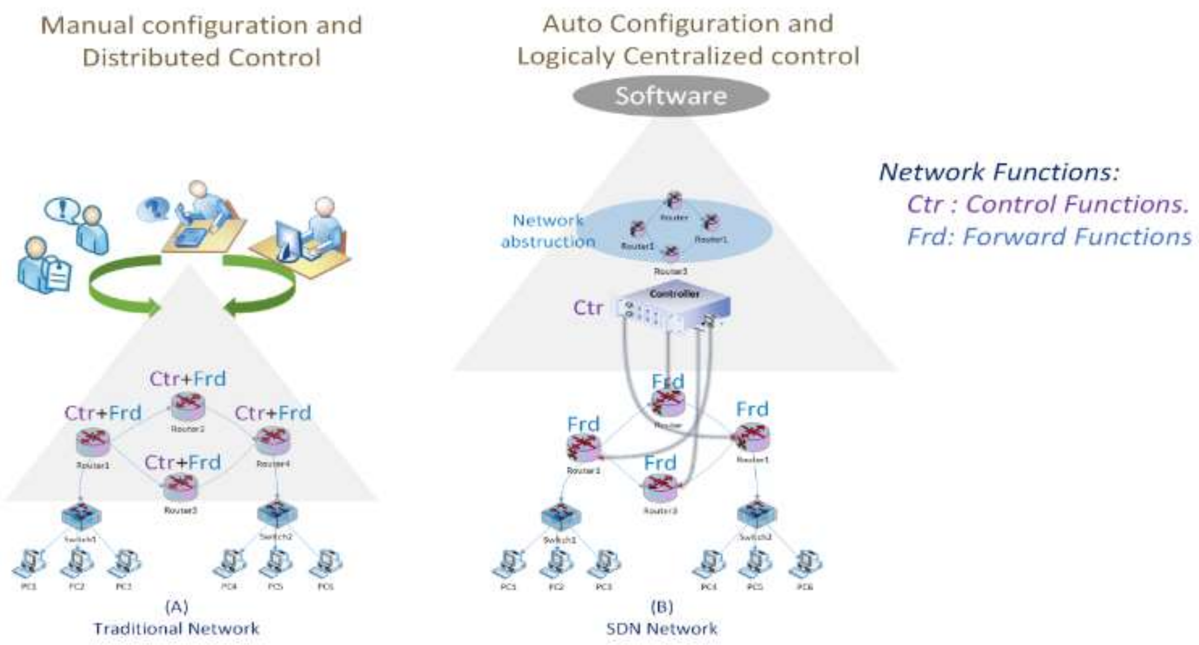


Figure 6: Traditional and SDN Network architecture (Al-Sadi, 2018)

The concept behind the vertical separation of the network architecture, as illustrated in Figure 6, aims to foster flexibility and innovation. This separation allows the control plane, integrated into the software configuration, to automatically manage the forwarding of hardware elements. Through the SDN controller, the application layers can be managed in a streamlined manner, simplifying the administration of network devices (Al-Sadi, 2018). This results in better agility for modifying network configurations, reduces the risk of errors, and speeds up the optimisation of network performance. A visual overview of software-defined networking and its advantages over the conventional model is shown in Figure 7.

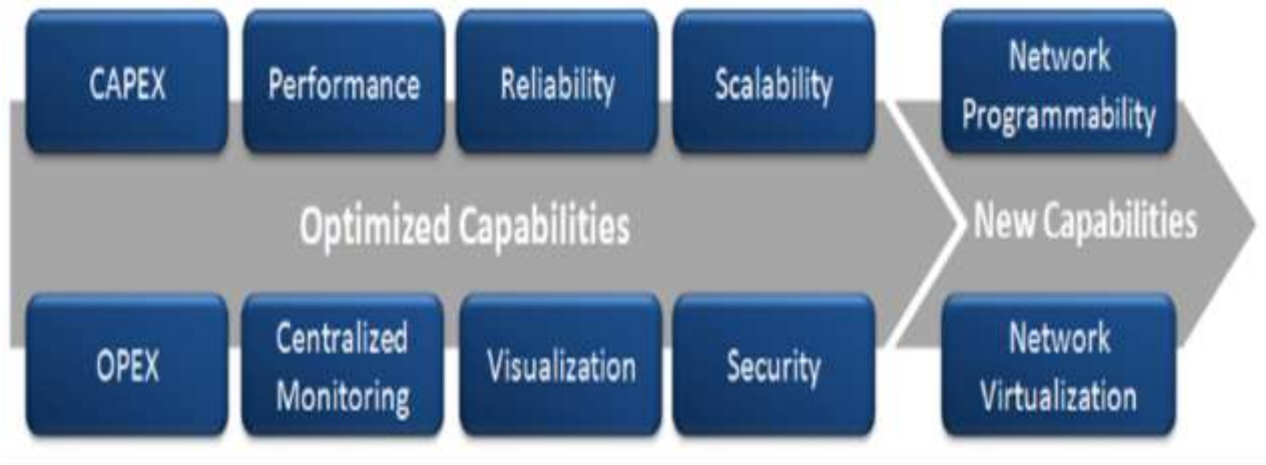


Figure 7: The benefits of SDN (Al-Sadi, 2018)

Figure 7: also depicts the numerous benefits of implementing the SDN architecture over the traditional networking model, including scalability, network visualisation, improved performance, centralised monitoring, reliability, security, and network programmability. The SDN architecture enables central provisioning, which allows for the centralised management of various networks by virtualising data and control plane elements in one location (Gkioulos et al., 2018; Keary, 2020; Liang & Li, 2018). This centralisation improves scalability and network capacity, which is not possible with a traditional architecture that requires the manual configuration of network devices/resources. Also, SDN deployment reduces the hardware footprint and automates device deployment. A centralised approach to security helps secure networks and prevent external threats, although this does make the SDN controller a potential target. But, it provides users with a clear understanding of their infrastructure and enables them to manage the security of their network effectively (Keary, 2020; Liang & Li, 2018).

2.3.1: Software Defined Networking Architecture

The SDN architecture comprises three planes, namely the Management plane, the Control plane, and the Data plane. The first layer, which is also known as the infrastructural layer, defines the Data plane and the forwarding elements. These forwarding elements refer to networking equipment such as switches and routers. They create the physical network infrastructure and are connected through wired connections or wireless radio channels. With an intelligent interface, they can communicate with upper layer devices via standardised interfaces called OpenFlow (Feamster et al., 2014).

In SDN architecture, the second layer is referred to as the control plane or controller platform. It consists of the network abstraction and controller programs and is responsible for linking the control plane and management plane using Northbound APIs. The Northbound APIs (Application Programming Interface) are essential in developing applications for network management, load balancing, and network slicing, as stated by (Ominike Akpovi et al., 2016). Figure 8: illustrates the various planes, levels, and structural design of software-defined networking.

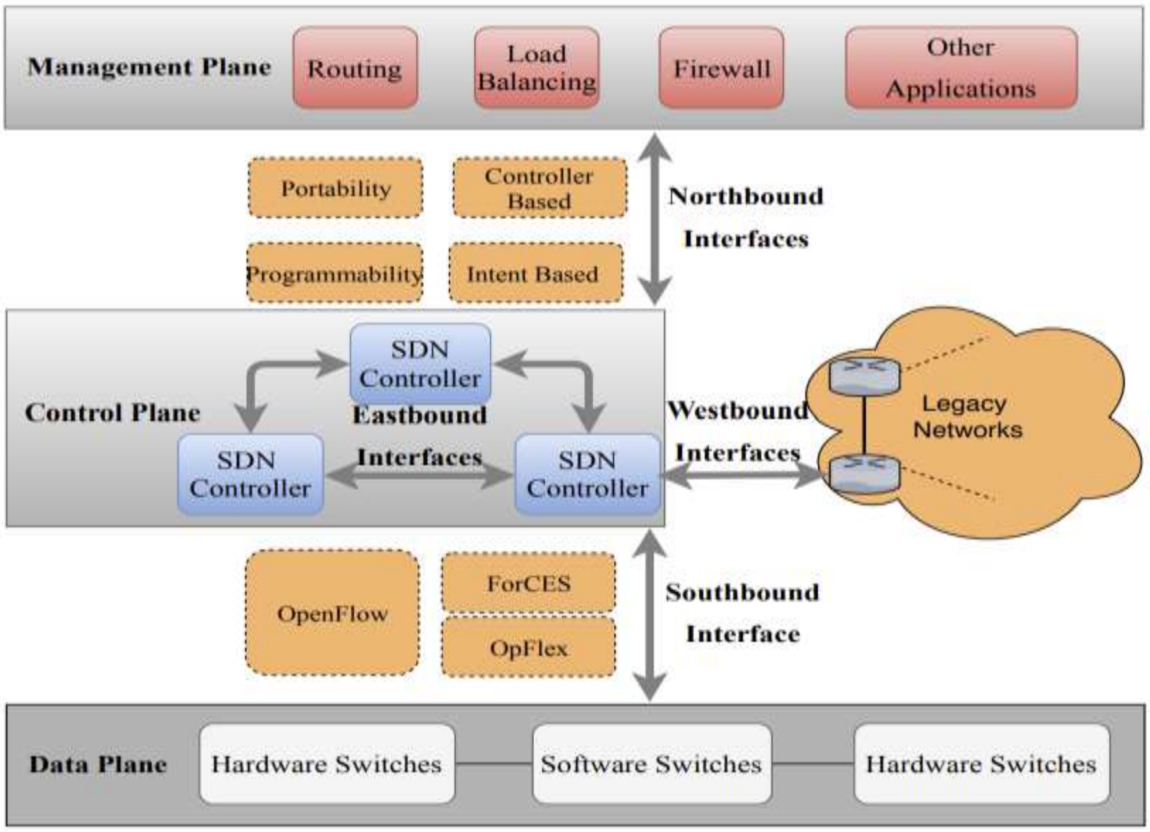


Figure 8: illustrates the various planes, levels, and structural design of software-defined networking.

2.3.2: Management Plane

The management plane is the application layer that manages the entire SDN architecture. It provides an abstract and global view of the network and offers real-time updates and guidance for critical applications within the network architecture. The management plane can introduce critical applications like security mechanisms, centralised control management, and monitoring applications to the network (Braun & Menth, 2014).

2.3.3: North-Bound Interface

Due to its programmability features, the North-Bound Interface (NBI) is a vital interface in SDN architecture. Unlike a protocol layout, it serves as a software platform that developers can utilise to create and maintain network resources. The NBI encompasses a range of network applications, controllers' relationships, and user applications or services, making it possible for the controller to interact with other systems and facilitate security orchestration. While there is not a universally accepted standard for the NBI, there are efforts to establish patterns and products that can help verify the development of a standard by the ONF (Open Network Foundation) (Kreutz et al., 2014; Latif et al., 2020).

Moreover, the NBI is designed to include fundamental network functions such as loop avoidance, path computation, security, and routing, as well as optimise performance in areas like load balancers, cloud service orchestration, and software-defined security. The NBI can be implemented in various ways, such as creating a general Linux control platform that allows administrators to define modules and control policies, creating ad-hoc Northbound APIs that are specific to the controller, or defining low-level applications that serve as southbound interfaces device drivers (Tijare & Vasudevan, 2016).

2.3.4: South-Bound Interfaces

The Southbound Interfaces play a role in the network control by facilitating real-time implementation of changes through the SDN controller. They operate as an interface between the control layer and the underlying networking equipment like switches and routers. The OpenFlow standard is the most used in the Southbound Interfaces/APIs, which was the first SDN standard API used for enforcing flow entries. However, standards like NETCONF (standardised by IETF), OpFlex (used by Cisco), and OF-Config (enabled by the Open Networking Foundation) are also available. Additionally, routing protocols like OSPF, IS-IS, and BGP are used to support the Hybrid operation of SDN (Hakiri et al., 2014).

In OpenFlow (OF) standard, network administrators can test new configurations and applications, split traffic, and ensure optimal performance for control flows. Many router and switch vendors, including Juniper, Cisco, Brocade, Big Switch Networks, IBM, Arista, Dell, HP, NoviFlow, and NEC, support the OpenFlow standard. OF provides a platform for researchers to test and experiment with innovative ideas, based on its capabilities for updated forwarding rules, traffic analysis, centralised control, and flow abstraction. In addition, the OF protocol uses three data transmission models within the network. The first model involves exchanging statistical information between the controller and forwarding components. The second model monitors the packets being forwarded to the controller from the networking components, while the third model ensures the delivery of flows related to the controller in case of any failure or changes in network conditions as shown in Figure 9 (Davie et al., 2017; Masoudi & Ghaffari, 2016).

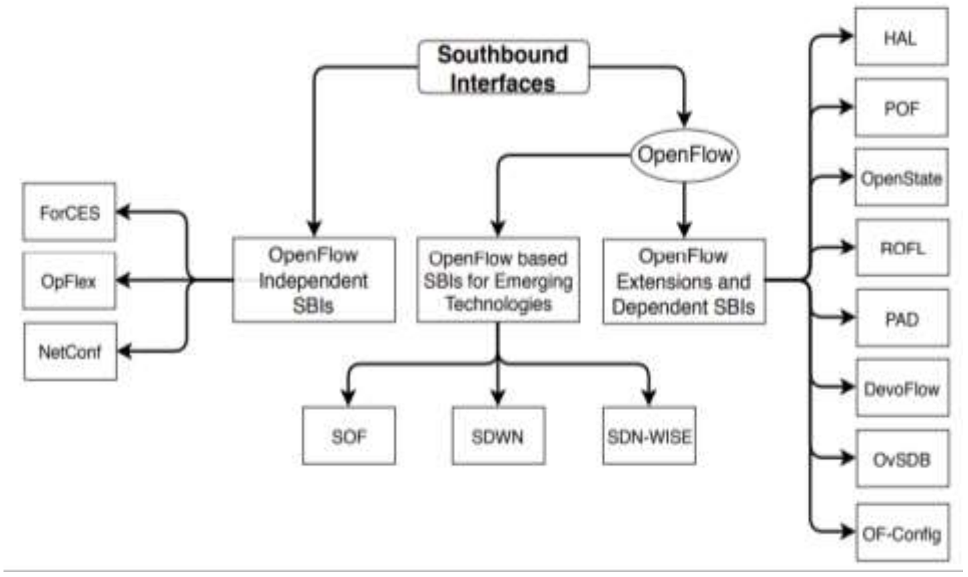

Southbound Interfaces
OpenFlow
ForCES
OpFlex
NetConf
OpenFlow Independent SBIs
OpenFlow based SBIs for Emerging Technologies
OpenFlow Extensions and Dependent SBIs
SOF
SDWN
SDN-WISE
HAL
POF
OpenState
ROFL
PAD
DevoFlow
OvSDB
OF-Config

Figure 9: South-Bound Interfaces (Latif et al., 2020)

2.3.5: East-bound/West-Bound Interfaces

The key principle of SDN is centralised control of the entire network. This approach has limitations as a single controller can only manage a limited number of switches. To address this issue, there is a need for a distributed controller model that can accommodate the growth of network devices and the scale of networks. In this framework, each controller is responsible for a set of network devices and infrastructure. The controllers have a global view of the network and share information consistently across all domains. To enable this sharing of information, the East-Bound Interfaces are used for exporting and importing network information between different distributed controllers. Meanwhile, the West-Bound Interfaces facilitate communication between the controllers and legacy networks. The West-Bound Interfaces also serve as a message conduit between SDN control planes in different network domains. This allows for the exchange of state network information to make routing decisions on each controller and enables the seamless setting up of network flows in multiple domains (Ji, 2023; Piech, 2023).

2.3.6: Control Plane (Controller)

The SDN architecture relies greatly on the controller, which serves as a crucial component responsible for managing the network traffic based on a set of predefined instructions, also known as flows. Within the SDN controller, there exist numerous functionalities, such as device management, notifications, statistics, and topology information. These functionalities are typically implemented using either North-Bound or South-Bound Interfaces. Controllers can be categorised as either centralised or distributed based on the network architecture. Centralised controllers use a single point of entry to manage the entire network infrastructure, whereas distributed controllers leverage a group of cooperating controllers to manage the underlying networking elements through the West/East-Bound interfaces.

There exist various controllers that are available for use in the architecture of SDN networks. Some of these controllers include POX, NOX, Floodlight, ONOS, RYU, and Open Daylight (ODL). It is worth noting that there are many more controllers that are used in SDN architecture. These controllers have been developed using a variety of common programming languages such as C, C++, Erlang, Go, Haskell, Python, and Java Scripts. Depending on the network conditions and resource utilisation requirements, some controllers have been built using a single programming language, while others have been built using multiple languages to achieve optimal performance. Most controllers are designed to run on Linux distribution platforms, with web-based or graphical interfaces provided for administrators. For lightweight SDN deployments and practical implementation, single-threaded controllers are suitable, while multi-threaded controllers are designed for commercial use in SD-WAN, optical networks, and 5G networks. Most of the controllers are open source and have online documentation, which ranges from fair to good quality. The controllers with good documentation typically have regular updates, along with community-based support for all versions (Karakus & Durresi, 2017; Zhu et al., 2019)

Table 1.0 summarises commonly used SDN controllers and their functions in terms of programming language, architecture, Northbound/Southbound API, licensing, Interface, and documentation. These controllers provide different features and functionalities for managing SDN environments, providing diverse requirements and preferences of users. In table 1.0, it is clearly established that various programming languages such as Java, C and Python are used in the implementation of the various controllers. Also, the documentation shows from fair to good with REST API as one of the dominant API used in the Northbound API.

Controller Name	Programming Language	Architecture	Northbound API	Southbound API	East-West Bound API	Licence	Interface	Documentation	Supported Platform	Modularity
Beacon (Erickson, 2013)	Java	Centralised	ad-hoc	OF 1.0	-	GPL 2.0	CLI, Web UI	Fair	Linux and Windows	Fair
Disco (Phemius et al., 2014)	C, JavaScript	Distributed Flat	REST	OF 1.0	AMQP	Proprietary	-	Limited	Linux	Good
Floodlight (Morales et al., 2015)	Java	Centralised	REST, Java & Quantum	OF 1.3, 1.0	-	Apache 2.0	CLI, Web UI	Good	Linux and Windows	-
Flow Visor (Sherwood et al., 2009)	C	Centralised	JSON RPC	OF 1.3, 1.0	-	Proprietary	CLI	Fair	Linux	-
Kandoo (Hassas Yeganeh & Ganjali, 2012)	C, C++	Distributed	Java RPC	OF 1.0-1.3	Messaging Channel	Proprietary	CLI	Limited	Linux	High
Loom (Kazarez, 2023)	Erlang	Distributed Flat	JSON	OF 1.3-1.4	-	Proprietary	CLI	Good	Linux	Good
Microflow (Zhang, 2023)	C	Centralised	Socket	OF 1.0-1.5	-	Apache 2.0	CLI, Web UI	Limited	Linux	-
Node Flow (Berger, 2023)	JavaScript	Centralised	JSON	OF 1.0	-	Cisco	CLI	Limited	Node.js	-
NOX (Gude et al., 2008)	C++	Centralised	ad-hoc	OF 1.0	-	GPL 3.0	CLI, Web UI	Limited	Linux	Low
ONIX (Koponen et al., 2010)	C++	Distributed Flat	Onix API	OF 1.0, OVSDB	Zookeper	Proprietary	-	Limited	-	Good
ONOS (Berde et al., 2014)	Java	Distributed Flat	REST, Neutron	OF 1.0	Raft	Apache 2.0	CLI, Web UI	Good	Linux, MacOS, Windows	High
Open Daylight (Zhu et al., 2019)	Java	Distributed Flat	REST, RESTCONF	OF 1.3 & 1.0	Raft, Akka	EPL 1.0	CLI, Web UI	Good	Linux, MacOS, Windows	High
POF Controller (Li et al., 2017)	Java	Centralised	-	OF 1.3 & POF-FIS	-	-	GUI, CLI	Limited	Linux	-
POX (Ryu, 2022)	Python	Centralised	ad-hoc	OF 1.3	-	Apache 2.0	GUI, CLI	Limited	Linux, MacOS, Windows	Low
RYU (Karakus & Durresi, 2017)	Python	Centralised	REST	OF 1.3	-	Apache 2.0	CLI	Good	Linux, MacOS	Fair

Table 1: Summary of commonly used SDN Controller Parameters

2.3.7: Data Plane (Network Infrastructure)

The data plane comprises the active networking devices such as physical/virtual switches, and routers. The consolidated interface ensures dynamic configuration of OpenFlow switches in SDN, which relies on programmable switches. These switches can communicate with the controller using the OpenFlow protocol, and comprise three main components: flow tables, secure channel, and the OpenFlow protocol. The secure channel establishes a connection to the controller, while the OpenFlow protocol facilitates communication with the external controller (Latif et al., 2020).

2.4: Software Defined Wide Area Network

SD-WAN, or Software-Defined Wide Area Network, is a hybrid deployment which consists of cloud-based or premise overlay architecture for wide area networks that operates based on the principles of software-defined networking. This technology is commonly used in enterprise networks to enable centralised control and secure the network intelligently across a WAN (Granath, 2023). SD-WAN is designed to ensure that critical enterprise applications receive predictable service, and that traffic is dynamically routed from one point of the WAN to another. Each SD-WAN system is equipped with SD-WAN devices that interconnect a WAN to a local area network and automatically download pre-defined network configurations and policies (Mehra, 2023).

SD-WAN utilises pre-defined network configurations and policies to enable dynamic path selection and traffic steering, optimising the quality of service and user experience for various clients and applications. This technology simplifies network architecture, enhances bandwidth efficiency, minimises costs, and guarantees secure cloud access with data privacy and security measures. Gartner has identified five primary components of SD-WAN, which include supporting multiple connections (such as MPLS, Internet, and LTE), dynamic path selection to distribute the load across WAN connections, virtual private networks, and third-party services (such as WAN optimisation), and a user-friendly interface for WAN management that facilitates zero-touch provisioning and simple configuration (Cooney, 2023a; Juniper-Networks, 2023b).

Figure 10: depicts a simplified comparison between traditional WAN and SD-WAN architecture. In the case of SD-WAN, it is possible to combine several transports, such as MPLS, broadband access, LTE (Long Term Evolution), and others. The SD-WAN controller can automatically switch between them or load balance the applications. In contrast, traditional WAN architecture is limited to a single transport service. Deploying SD-WAN enables the use

of more affordable public ISP and provides customers with newer and better services at a lower cost of using MPLS VPN services only (Sabiq et al., 2023).

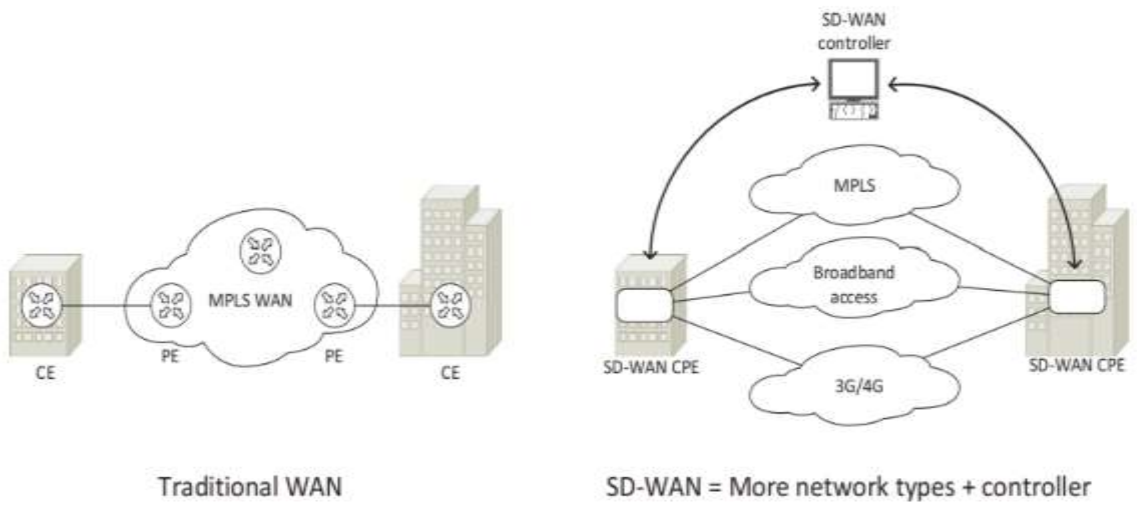


Figure 10: Traditional WAN and SD-WAN settings

2.4.1: Architecture of Software Defined Wide Area Network

The SD-WAN architecture comprises an SD-WAN controller, multiple transport networks and SD-WAN gateways that connect between sites. SD-WAN architecture can be further classified into a physical and logical structure. Each of the components used in an SD-WAN architecture has a significant role in the operation of the entire network. The architecture of the SD-WAN is presented in Figure 11.

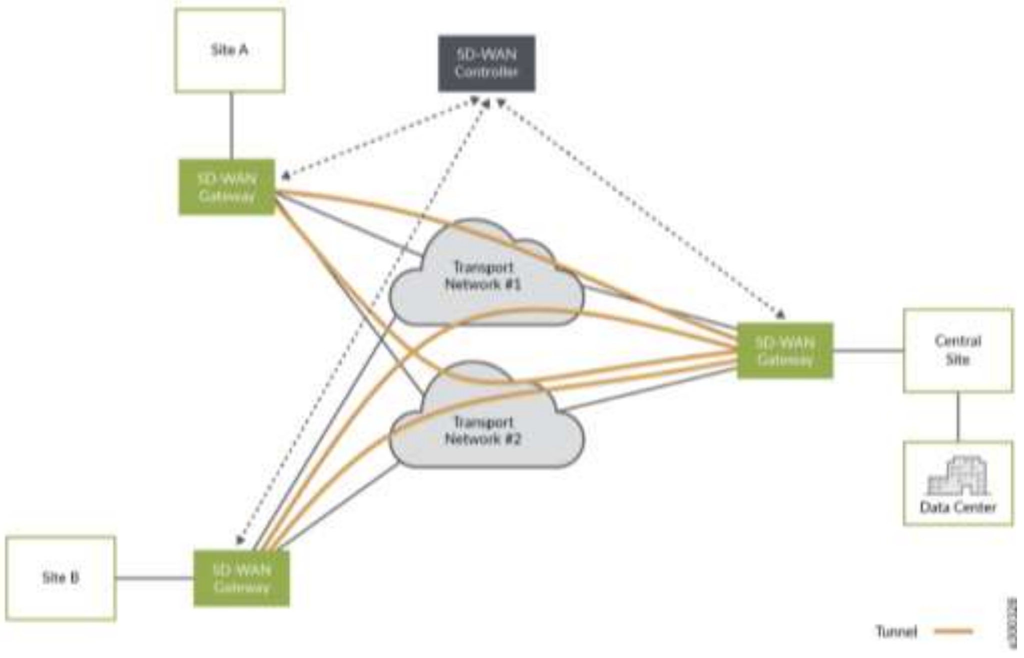


Figure 11: Operational architecture of SD-WAN (Jensen, 2018)

2.4.2: Logical Architecture

The logical structure of the SD-WAN architecture can be described easily using Cisco proprietary devices as described in Figure 12. The cisco proprietary devices represented are vBond, vManage, vSmart and vEdge.

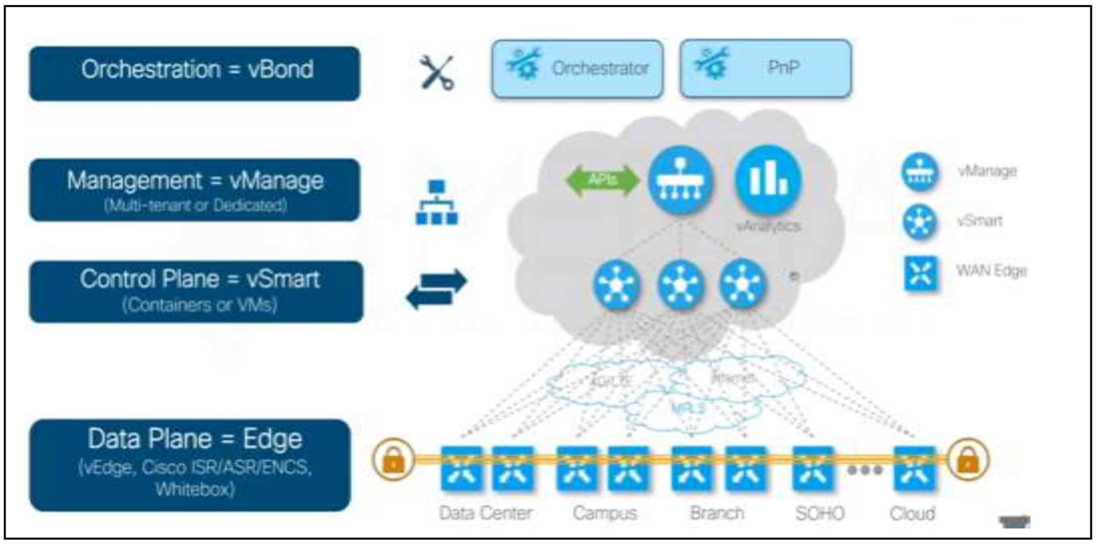


Figure 12: Logical architecture of SD-WAN (Jensen, 2018)

SD-WAN as a cloud-delivered service consists of four planes, these include the data plane, control plane, management plane and orchestration plane. These are then mapped onto four main components called vEdge, vSmart, vManage and vBond respectively. The forwarding of traffic in the SD-WAN architecture's data plane is handled by vEdge switches and routers. These vEdge devices establish a secure plane connection to the vSmart controllers, which manage the control plane responsible for enforcing rules and policies that determine how various sites communicate with each other. The vSmart controllers ensure they pass accurate information to the vEdge devices and when the information is not consistent and identical, it will result in network connectivity issues. The vManage component of the SD-WAN architecture represents the user interface in the application layer, including the application server, configuration database, network configuration, message bus, and statistics database, all of which are integrated into the architecture for optimal functioning. The vManage provides a single dashboard for multiple customers or users, enabling easy deployment and management. The orchestration plane is managed by the vBond component, which performs zero-touch provisioning, authentication, information management/control, and Network Address Translation (NAT) implementation. The vBond component is responsible for onboarding devices to the SD-WAN fabric, and multiple vBonds are present in each SD-WAN network architecture(Franjić, 2023; Sabiq et al., 2023; Z. Yang et al., 2019).

2.4.3: Physical Architecture

The physical architecture of SD-WAN consists of three layers: data, control, and application layer as illustrated in Figure 13.

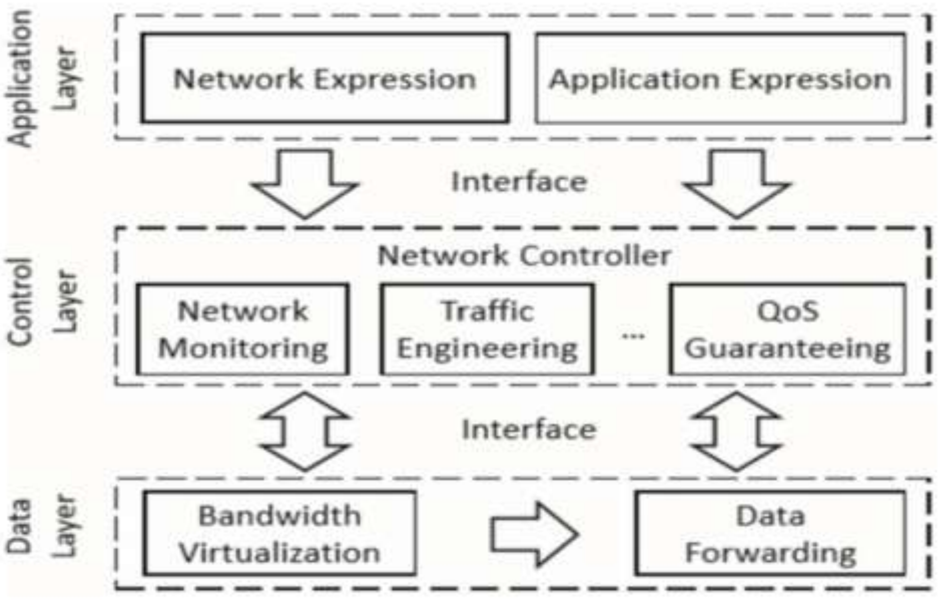


Figure 13: Physical Architecture of SD-WAN (Yalda et al., 2022)

The data layer is composed of interconnected devices like routers and switches that are linked through physical connections. These devices forward the traffic flows or transmit packets. The network control layer can be a server or a cluster, depending on the size and complexity of the network, and encompass networking monitoring tools, quality of service provision, and traffic engineering (Bannour et al., 2017; Jain et al., 2013). The network controller manages various functions and applications, communicates application requirements, and translates them into compatible configurations (Yalda et al., 2022). The application layer allows network and application developers to collaborate with network controllers to implement policies for different applications that are transported through the network. As applications often have complex and varied requirements, it is necessary to modify network strategies to accommodate them, hence the need for the application layer (Hartert et al., 2015; Yin et al., 2015).

2.4.4: Underlay and Overlay (SD-WAN)

The physical connections between devices in the SD-WAN setup make up the underlying network. This network layer does not have any knowledge of the LAN segments belonging to the customer, and its purpose is only to establish connectivity between on-premises devices. To achieve this, packet encapsulation is used at layer 2 (switches) and layer 3 (routers). The TLOCs (Transport Locators), represented by coloured lines (Green and Orange) in Figure 14, serve as connection points to the transports and are vital in separating the underlay network from the overlay fabric and applications. The sole function of the underlying network is to enable IP reachability between the TLOCs. For the underlay network to function properly, the WAN interface connection requires an IP address, an encapsulation type, and a colour. These parameters are communicated to the controllers through OMP (Overlay Management Protocol), which is part of the TLOC route advertisements (Juniper-Networks, 2023a; Network-Academy, 2023).

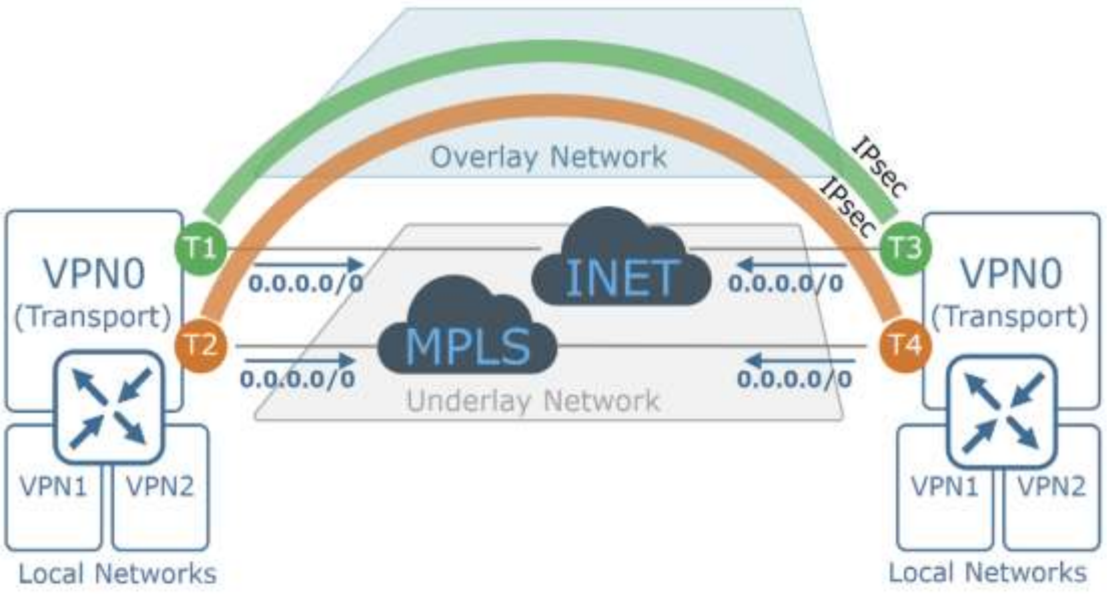


Figure 14: Description of the underlay network (Juniper-Networks, 2023b; Network-Academy, 2023; Nicholas, 2023).

The overlay network consists of the logical tunnel connectivity between the various devices in the SD-WAN architecture and has awareness of the LAN segment and is tasked in transporting the customer traffic between sites. In a Cisco implementation, routing is implemented by the Overlay Management Protocol (OMP), which is like the BGP (Border Gateway Protocol). The OMP protocol operates TLS (Transport Layer Security) for TCP connections or over secure DTLS (Datagram Transport Layer Security) for UDP (User Datagram Protocol) between vSmart controllers and edge routers. The vSmart controller functions as a BGP route reflector (RR), receiving, modifying, and re-advertising routes from the vEdge routers, without participating in the data-plane or packet forwarding (Juniper-Networks, 2023b; Network-

Academy, 2023; Nicholas, 2023). Figure 15: shows the relationship between the underlay and overlay network indicating the various links.

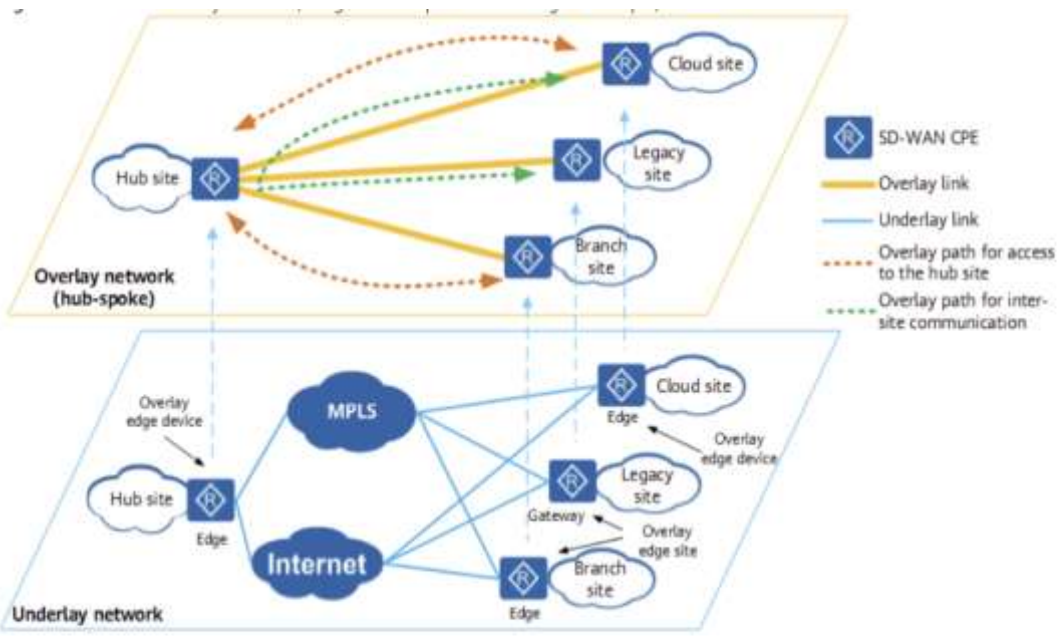


Figure 15: Description of the overlay network (Heng, 2018)

Figure 15 illustrates the overlay network connections, distinguishing them through different line styles and colours. The solid blue lines depict the connections within the underlay network, while the dashed blue lines indicate the connectivity between devices in the underlay and overlay networks. The solid gold lines represent the overlay links, connecting the cloud, legacy, and branch sites, while the dashed orange line represents the overlay path for accessing the hub site. In Figure 15, an SD-WAN network comprises customer-premises equipment (CPEs) categorised as edge devices and gateways. The edge device functions as the egress point for an SD-WAN site, while the gateway serves as the connecting device between SD-WAN sites and external networks, such as legacy VPNs. Meanwhile, the legacy site refers to an existing branch location that has not been migrated to SD-WAN and the branch site typically is the location where SD-WAN equipment is installed and configured to leverage SD-WAN technology for network management and connectivity. Additionally, the dashed green lines describe the overlay path used for inter-site communication within the overlay architecture (Heng, 2018).

2.5: SD-WAN Load Balancing

When wide area networks are connected through one or more links, ensuring reliable performance can be a challenge due to the unpredictable nature of network infrastructure, even with redundancy measures in place. As illustrated in Figure 16, traffic distribution between the enterprise network and the internet cannot be guaranteed, creating the potential for congestion of traffic flows because of packet loss or changes in the network conditions. Hence, there is a pressing need to balance traffic flow and load between end-to-end stations to ensure even distribution and avoid congestion.

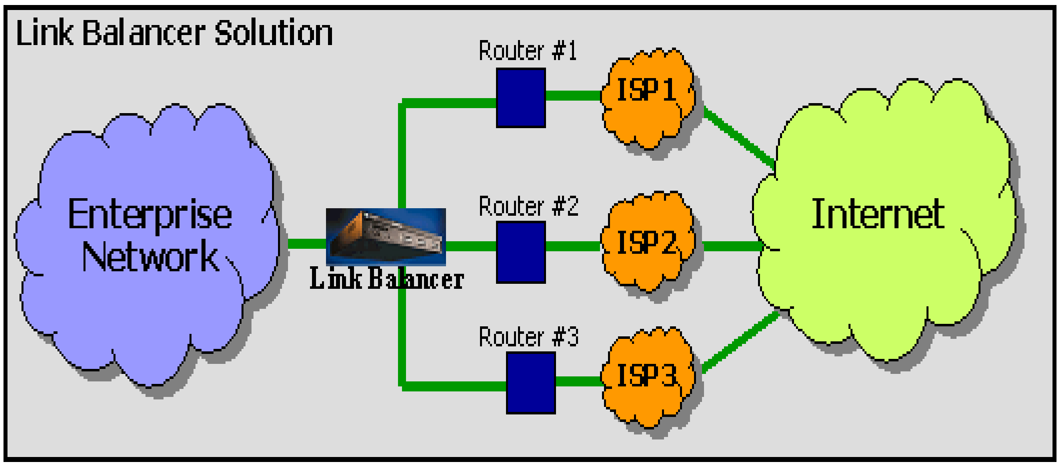


Figure 16: Load Balancing Illustration for Different ISP in a Network (source:(Rouse, 2019))

Business-critical applications have been impacted by limited bandwidth and inadequate sharing of network resources. The lack of business-oriented WAN scalability, wherein a single internet circuit is utilised for site-to-site connectivity, can lead to unpredictable network behaviour (Q-Balancer, 2023). To address these uncertain WAN issues, load balancing mechanisms are proposed to effectively prioritise critical applications and scale the network using recent technology such as SD-WAN (Ivanisenko, 2015). There exist numerous load balancing algorithms, and the specific technique or algorithm used depends on the network demand or status (Davies, 2022). These algorithms vary considerably depending on the load being distributed at the network or application layer. While application and network layer algorithms differ slightly in performance, they are both affected by the shared load between networks(NRaghava & Singh, 2014).

Server Load balancing algorithms can be categorised into static and dynamic algorithms. In static load balancing algorithms, load distribution is not influenced by the current state of network performance or resources. Dynamic load balancing algorithms are designed to be flexible and leverage prior knowledge of the network's current state (Deepa & Cheelu, 2017). Therefore, implementing an effective load balancing methodology in an SD-WAN architecture can improve system performance, ensure system stability, and make the system more resilient to faults and accommodate new network policies. Figure 17: illustrates some of the load balancing models employed in the transport and application layer, while Table 2 presents a comprehensive review of different load balancing mechanisms based on their methodology, real-time application, and load distribution patterns.

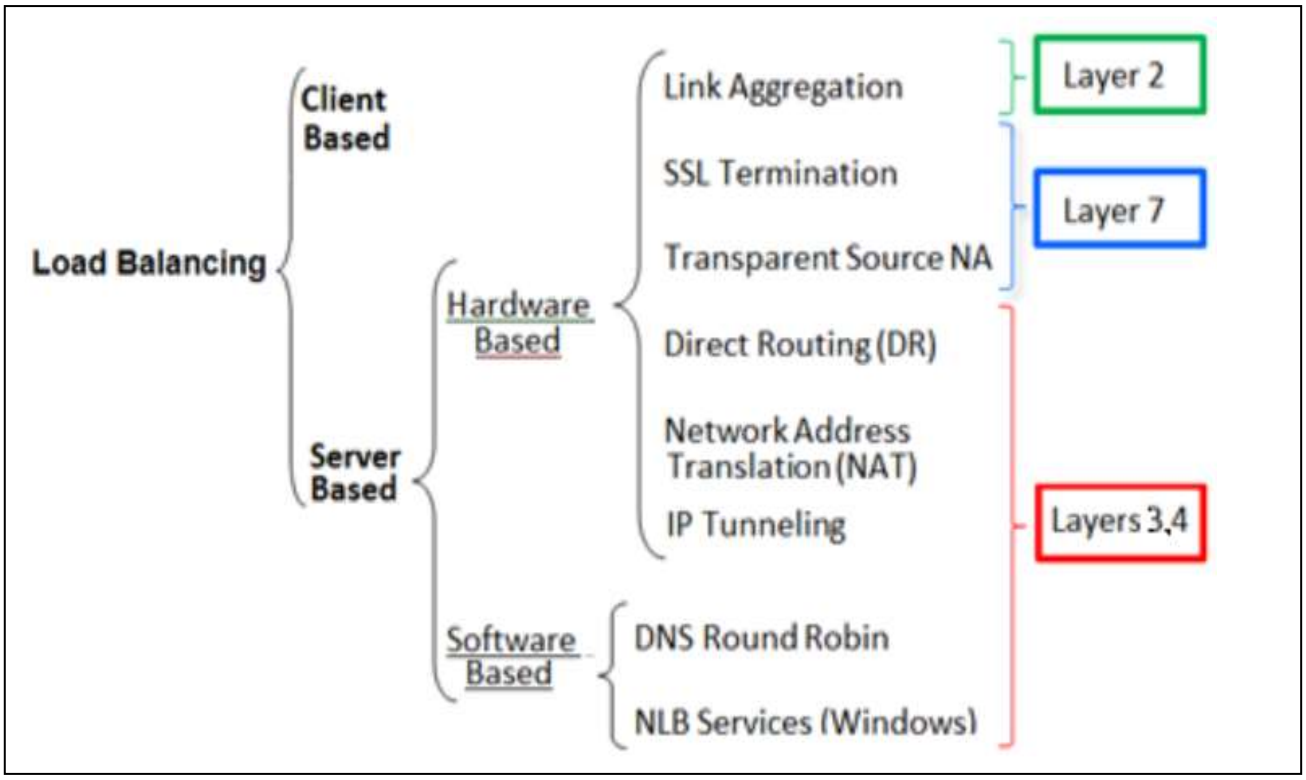


Figure 17: Server Network Load Balancing Model for the OSI Model (Ivanisenko, 2015)

Figure 17 displays different load balancing techniques categorised into software-based and hardware-based approaches. The software-based load balancing methods include round robin and network load balancer. The hardware-based techniques encompass link aggregation, direct routing, Network Address Translation (NAT), and IP tunnelling.

Load balancing algorithms	**Methodology**	**Real time application**	**Server load distribution**
Round Robin Algorithm	A circular distribution of traffic requests to enterprise servers following a sequence. This is implemented using time slices where each given quantum performs its operations.	The dynamic round robin mechanism can used real time calculation based on assigned server load while the static round robin does not.	Servers are rated based on the relative amount of request.
Least Connections	This is a network-based algorithm where requests are sent based on the fewest number of active connections in the networks.	Least connections sent requests based on the active/inactive connections but not based on real time application.	Loads are distributed taking account of the relative capacity of the servers and number of active connections.
Throttled Load Balancing Algorithms	This algorithm operates effectively in virtual machines. The clients check the available and right virtual machine to access the load and perform its operations.	Maintains an index of available virtual machines with their various states.	The load distribution pattern depends on the first server that makes a request and connects to a virtual machine.
Source IP hash	Operates by generating a hash key that combines with the source and destination IP which is sent to a specific server.	Based on pre-defined rules and generation of hash keys.	Based on generated hash keys, servers are assigned to various clients.
Least bandwidth	This involves the selection of servers based on the bandwidth consumption of the backend servers.	Based on pre-defined rules of server bandwidths measured in Mbps.	Based on the consumption of Bandwidth load has been distributed.
Fixed Weighting	Fixed weighing is the process by which an administrator assigns traffic or load to various applications server in a network	The administrator assigns and control the traffic based on their chosen	Application with highest demand takes all the bandwidth, if it fails the next highest takes all the bandwidth
Least Pending Request	This is an application-based load balancing algorithm that monitors HTTPS requests and distributes them to the most available server application.	The features required dynamic approach based on real-time application	HTTPs traffics are distributed to available server applications based on request
Load balancing algorithms	**Methodology**	**Real time application**	**Server load distribution**

Dynamic Load Balancing Algorithm	Use a dynamic approach for effective network congestion and balancing of the load but lots of overhead and Quality of Service is not considered.	Operates in real time because of the dynamic nature of the algorithms.	Loads are distributed amongst various servers.
Quality of Service Load Balancing Algorithms	Load balancing is implemented in long term evolution networks with the aim of reducing overload and improving network performance.	Operates in adaptive nature.	Loads are distributed amongst various nodes.
Controller Placement Load Balancing Algorithms	This involves sharing the load amongst various multiple controllers for the reduction of load difference and migration cost.	Based on predefined rules.	Load is distributed amongst multiple controllers for better load balancing.
Direct routing-based Algorithm	This involves even distribution of the load with the aim of minimising latency in the network and better throughput.	Operatives in adaptive nature.	Load is distributed among controllers better with a single controller.

Table 2: Review of different load balancing mechanisms based on the methodology, real time applications and load distribution pattern.

The information in Table 2 provides an overview of various load balancing algorithms and their methodologies in real-time applications. These algorithms offer different approaches to distribute load among servers, considering factors such as server capacity, connection count, bandwidth consumption, and application-specific requirements. Understanding these load balancing algorithms can help optimise network performance and improve resource utilisation in diverse environments in the SD-WAN architecture.

2.6: Packet Inspection Technology

There are several technologies for carrying out deep packet inspection such as NetFlow, Content Filtering, Application Performance Management and Network Behaviour Analysis. NetFlow is a networking tool that is used in capturing and analysing packets developed by Cisco. It gives insight to the network traffic pattern with the aim of troubleshooting, performing security analysis and network planning to resolve network congestion and improve performance. One of the limitations of the NetFlow tools is a Cisco Proprietary device which does not allow the provision of open source and it generates a lot of network traffic but is not able to detect key security threats like encrypted traffic when applied (So-In, 2009). The Application Performance Management (APM) tool is used in monitoring and managing various network applications in real time. Some of the metrics monitored includes response time, transaction volume and resource usage/aggregation. Based on the resources, it can track or trace network flows from one point to another which helps in resolving performance issues, capacity planning and increase of quality of experience. However, there are limitations to the use of Application Performance Management when applied to a complex distributed environment and the problem of analysing large chunks of data (Aldossary, 2021). State Packet Inspection (SPI) operates by using the state of each connection passing a router or firewall to determine which packets can be blocked or allowed within the threats level. The SPI enhances network performance, increased flexibility, and enhanced network security. However, there are limitations with the implementation because of the resource-intensive methods which slow down network performance when not implemented correctly. The SPI implementation focuses only on the network layer and may be exposed to certain attacks such as spoofing. Thus, the effectiveness of SPI depends on the rule that is associated or designed to operate it (Xu et al., 2016).

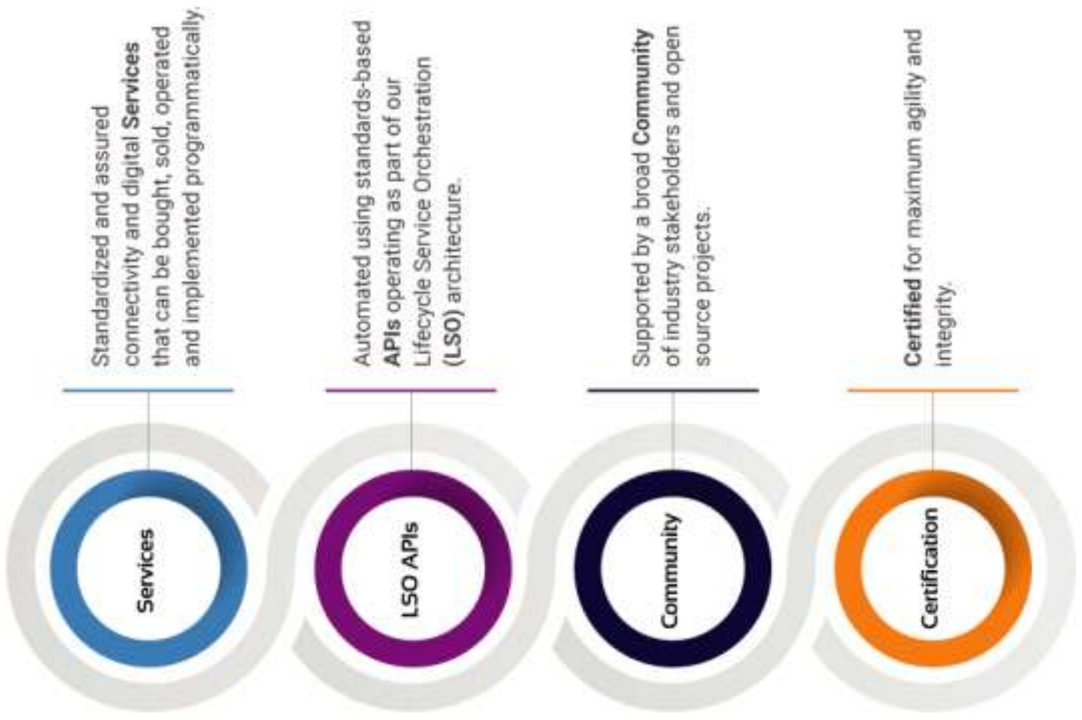
Network Behaviour Analysis (NBA) uses statistical tools and machine learning algorithms to detect and identify anomalies in network traffic which indicate security threat. NBA specifically focuses on the network intrusions, malware, data exfiltration and insights into the network patterns which is used in optimising network performance and overall security architecture of the network. Furthermore, (Orans et al., 2020) suggested NBA is effective in identifying a broad range of threats that traditional security technologies, such as firewalls and intrusion prevention systems, may miss. However, the same study also notes that NBA is not a silver bullet and should be used in conjunction with other security technologies to provide comprehensive protection.

Moreso, it is a content filtering technology that has evolved over the years and is used in major sectors to enforce acceptable policies that are allowed in organisations. An effective content filtering policy ensures better regulatory compliance, increased productivity and by enforcing content-related policies. However, content filtering has its limitations which is the difficulty in appropriating and non-authorisation of some network traffic. According to (Ponemon Institute, 2019): Content filtering is an effective means of upholding acceptable use policies and preventing unauthorised or inappropriate content access. However, it is crucial to complement content filtering with additional security measures like network monitoring and access control to ensure comprehensive protection.

The Deep Packet Inspection (DPI) tool is a network security technology that examines network traffic at the packet level with the purpose of identifying the content, protocols, and applications in real time. It involves key traffic monitoring policies such as traffic shaping, monitoring, network security and behaviour performance tools (Parra et al., 2019). DPI is very effective in improving the quality of service, network performance optimisation, detect possible network intrusion and prevent any form of data exfiltration. Though, it is limited because of the intensive use of resources leading to increase in delay and latency in the network infrastructure. For this research deep packet inspection was chosen as the preferred method for data collection by the traffic monitor. The selection of this model was influenced by the limitations identified in other presented models, which did not offer a comprehensive solution to all concerns regarding packet capturing and analysis.

2.7: SD-WAN Standardisation

Both end users and vendors recognise the significance of certification or standardisation in technology, and SD-WAN is not an exception. To ensure compliance with policies, security measures, and functionalities, various service providers and end users are interested in standardisation (Corn, 2023). MEF (Metro Ethernet Forum) is a leading organisation that focuses on certifying or standardising the SD-WAN operational model. The organisation consists of global industries of network equipment/software manufacturers, telecommunication service providers, and testing organisations, effectively defining dynamic services of the SD-WAN operation. MEF's first proposed standard for SD-WAN is MEF 3.0, which outlines four key components of the SD-WAN model: services, LSO (Lifecycle Service Orchestration) APIs, community, and certification (ONF, 2023). Figure 18: provides a graphical summary of MEF 3.0's SD-WAN components.



Standardized and assured connectivity and digital **Services** that can be bought, sold, operated and implemented programmatically.
Automated using standards-based **APIs** operating as part of our Lifecycle Service Orchestration **(LSO)** architecture.
Supported by a broad **Community** of industry stakeholders and open source projects.
Certified for maximum agility and integrity.
Services
LSO APIs
Community
Certification

Figure 18: SD-WAN Components Described by MEF 3.0 (Cohn, 2019).

MEF 3.0's advancement has resulted in the creation of the MEF 70 standard, which introduces fresh service features for the underlay network connectivity services in the SD-WAN architecture. This upgrade is analysed by exploring novel performance metrics that offer insight into application performance across various service providers and provider networks, with the goal of enhancing application security within the network (Power, 2023). Figure 19: presents the service elements for SD-WAN as defined by MEF 70.

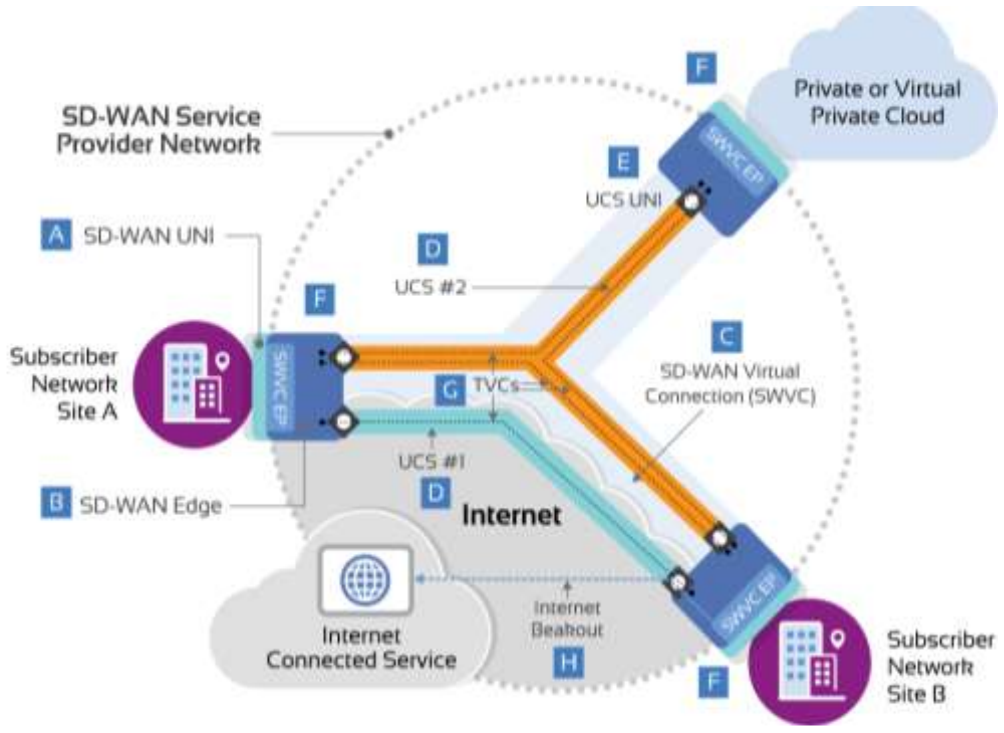

SD-WAN Service Provider Network
Private or Virtual Private Cloud
F
E
UCS UNI
SWVC EP
A SD-WAN UNI
D
UCS #2
F
C
Subscriber Network Site A
SWVC EP
G TVCs
SD-WAN Virtual Connection (SWVC)
UCS #1
D
B SD-WAN Edge
Internet
Internet Connected Service
Internet Breakout
H
SWVC EP
F
Subscriber Network Site B

Figure 19: Comprehensive Certification Components for SDWAN (Bob Victor, 2018)

The MEF 70 standard defines the various services within the SD-WAN network components, which are outlined in the service components. These components comprise the SD-WAN User-to-Network Interface, SD-WAN Edge, SD-WAN Virtual Connection, Underlay Connectivity Service, UCS User-to-Network Interface, SD-WAN Virtual Connection End Point, Tunnel Virtual Connection, and Internet Breakout(Victor, 2023). An updated version of MEF 70 is MEF 88, which was introduced in 2021 and focuses on the security functions and standards of SD-WAN. These standards include: the definition of malware protections, threats, security policy terminology, and actions to be taken in the event of a threat (MEF88, 2023). MEF 88 standards encompass middle-box functions (such as encryption and decryption), IP, port, and protocol filtering, malware detection and removal, domain name protocol, and URL filtering (MEF88, 2023). The MEF 95 standard for SD-WAN, introduced in 2022, introduces a policy framework, network slicing, and SASE. This standardisation effort emphasises the significance of SD-WAN in the industry and ensures a level playing field for all players. Furthermore, it fosters the exploration of new ideas, innovations, and concepts, while also providing guidance to researchers on areas where they can contribute to the development of SD-WAN products.

2.8: Summary

A comprehensive understanding of the importance, architecture, and operational aspects of SDN and SD-WAN has been gained through the background review. In the following chapter, the focus will shift towards academic literature and research conducted within the industry.

CHAPTER THREE: INDUSTRY AND ACADEMIC REVIEW

This chapter describes the academic and industrial viewpoints of SDN and SD-WAN technology, specifically examining their relevance to vital network functions such as load balancing, scalability, and controller placement. One section provides a comprehensive analysis of the academic research presented, while others focused on the industrial outlook of the technology.

3.1: Overview

The transition from conventional networking to software-based networking has attracted the interest of numerous researchers due to the benefits it offers. Researchers from both the academic and industrial sectors are interested in refining the established concepts to meet customers' quality of experience requirements with minimal operational and capital expenditures. As many companies invest in this innovative technology, it is vital to evaluate past and future work that could improve its performance (Cooney, 2023b).

3.2: Congestion Control Algorithm

A key function of the traffic monitor is to detect when a given traffic flow either enters or exist a congested state. Hence, it is essential that the traffic monitor can determine this from an inspection of the packets comprising each traffic flow. Taking TCP as an example, that protocol uses several techniques to manage congestion. These techniques include slow start (SS), congestion avoidance (CA), fast retransmit, and fast recovery (Allman et al., 2009). Slow start is utilised to probe the link's capacity and gather information about its characteristics, while congestion avoidance aims to prevent network congestion and adapt to changing network conditions. Fast retransmit and recovery are effective mechanisms for promptly retransmitting lost packets and recovering from transmission losses.

Congestion in network communication is monitored through metrics such as packet loss, latency, and rapid changes in window size. Packet loss occurs when network congestion leads to dropped packets, while latency increases with congestion, indicating delays in data transmission. The rapid changes from a large window size to a smaller one could potentially signify congestion within the network.

This research monitors each traffic flow to examine flags within packet headers to identify evidence of congestion as summarised in Table 3. In view to analyse and identify these congestion flags, it is important to examine and analyse various congestion control algorithms based on their methodology, congestion detection mechanism, and changes in TCP/IP parameters within the network, as illustrated in Table 3.

Algorithms	Methodology	Congestion Detection	Action/Recovery	TCP/IP Parameter Changing
TAHOE (Stoica, 2005)	It uses a go-back-n model with a cumulative acknowledgment. It involves the use of Slow-Start, Congestion avoidance and fast retransmission.	It uses Timeouts to detect packet loss and packet loss is used as a sign of congestion.	Uses Fast retransmission algorithm where the sender sets the SSTHRESH to half the current value CWND.	Window size, Sequence and ACK number
NEWRENO (Floyd et al., 2004)	Slight modification of the RENO and has the same principle as TAHOE but adds intelligence to it for early packet detection.	Uses congestion avoidance to detect congestion via a single RTT. It causes for the network to have congestion, when 3 duplicates ACK is received by the sender.	Uses fast retransmit by setting SSTHRESH to half window value and linearly increases CWND by 1 for any ACK received.	Window size, Sequence and ACK number
VEGAS (Srijith et al., 2005)	Intelligently apply AIMD by increasing MSS by at most one for every RTT	Observation of Packet loss, 3 duplicates ACK or timeouts in detecting congestion in the network.	Applies a modified Slow-Start algorithm to effectively control the CWND by increasing or reducing windows based on the current value of RTT. Halved the CWND value	Window size and RTT
BIC TCP (Ha et al., 2008)	Grow window at midpoint between the last window size with a packet loss (max) and the other window size without a lost packet (min)	Packet drops or loss in the network, delay-based approach	Involves the use of slow increase around Wmax and linear increase during additive increase and max probing to quickly get rid of congestion or packet loss in the network.	Window size, Sequence and ACK number
CUBIC (Ha et al., 2008)	Improved version of BIC, uses concave and convex approach	It is a loss-based algorithm; thus, congestion has been detected when there is a loss or packets drop, also timeout plays a key role	Involves the use of the concave and convex cubic functions, concave is where the window quickly ramps up the window size before the congestion portion and convex is where CUBIC probes for more bandwidth slowly at first then very rapidly.	Window size, Sequence and ACK number
TCP DUAL (Durand et al., 2011)	Focused on the reduction of CWND dynamic in CA mode of TCP Tahoe.	It estimates the queuing delay by using RTT measurement to indicate network congestion.	Half the current value of CWND and begins a slow start mode with an initial window of 1.	RTT Measurement

Algorithms	Methodology	Congestion Detection	Action/Recovery	TCP/IP Parameter Changing
TCP – FAST (Wei et al., 2006)	Involve the use of fixed number of packets in the bottleneck queue by using RTTbase and current average of RTT	low queue delay, packet loss	recover using the mathematical expression for window queueing using the variables of queueing smoothing factor.	RTT Measurement
TCP-NICE (Venkataraman i et al., 2002)	Operates by counting the number of times that the estimated bottleneck queuing delay is greater than a fraction of the max queuing delay.	awareness of packet loss, 3 duplicates ACK or timeouts in detecting congestion in the network.	If count is greater than fraction, half the CWND and do the same when loss is detected. Threshold defines the targeted fraction.	Window size and RTT Measurement
TCP-LP (Kuzmanovic & Knightly, 2006)	Uses timestamps of the max and min between sender and receiver	awareness of packet loss, 3 duplicates ACK or timeouts in detecting congestion in the network.	Determine the one-way queueing delay mathematically, based on that halved CWND for 1st congestion and at 2nd congestion, set CWND to one MSS	Window size and RTT Measurement
PERT (Al-Saadi et al., 2019)	Uses a probabilistic back off algorithm in delay measurements and smooths instantaneous	awareness of packet loss, 3 duplicates ACK or timeouts in detecting congestion in the network.	Probabilistic back off algorithm mathematical model is applied to threshold value, especially in the 0.65 multiplication decrease	Window size
LEDBAT (Shalunov et al., 2012)	Is a low extra delay background transport CC to monitor file transfer plus peer-to-peer network.	Monitoring of queuing delay as an early signal of network congestion	Use mathematical expression to determine the congestion in the window queueing with growth gain and threshold target as variables	Window size
TIMELY (Mittal et al., 2015)	Uses a rate-based gradient to ensure high accuracy via raw measurement of RTT in two thresholds	awareness of packet loss, 3 duplicates ACK or timeouts in detecting congestion in the network.	Use mathematical expression to determine under and over utilised network congestion, such as rate and normalised gradient	Window size and RTT Measurement

Table 3: Summary of Congestion Control Algorithm

The presented table 3, highlights changes in window size, sequence number, and acknowledgment number are among the changing parameters that signify congestion in the network. Also, the measurement of round-trip time (RTT) serves as a changing parameter that indicates network congestion.

3.3: Industrial and Market Context

The industrial viewpoint relates to the prominent entities in the SD-WAN market who play an active role in designing, configuring, and implementing SD-WAN solutions for diverse users. According to a report by the IDC (International Data Corporation), the market base for SD-WAN solutions is predicted to expand and witness a surge in the number of users from 2020 onwards (Nangare, 2023). Figure 20: showcases the key players in the SD-WAN deployment industry as analysed in the report.

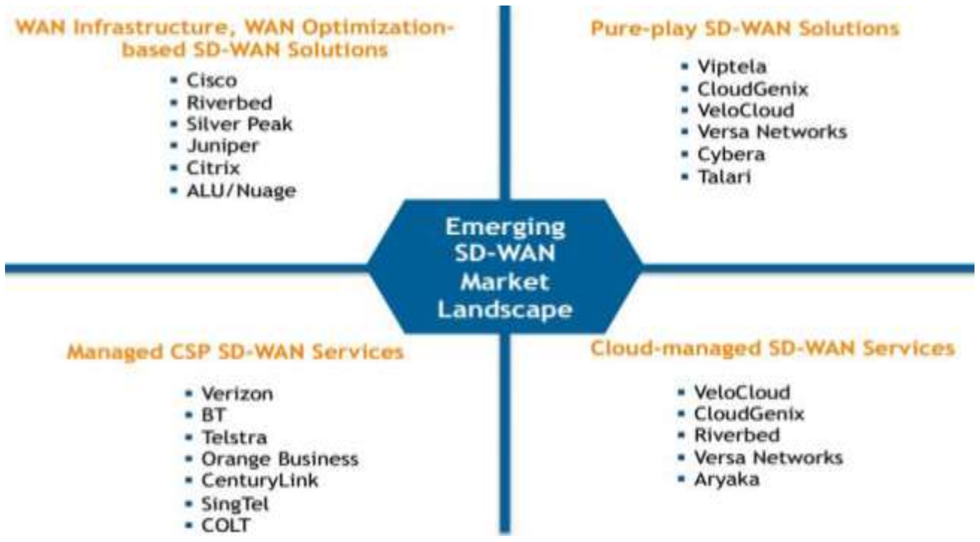
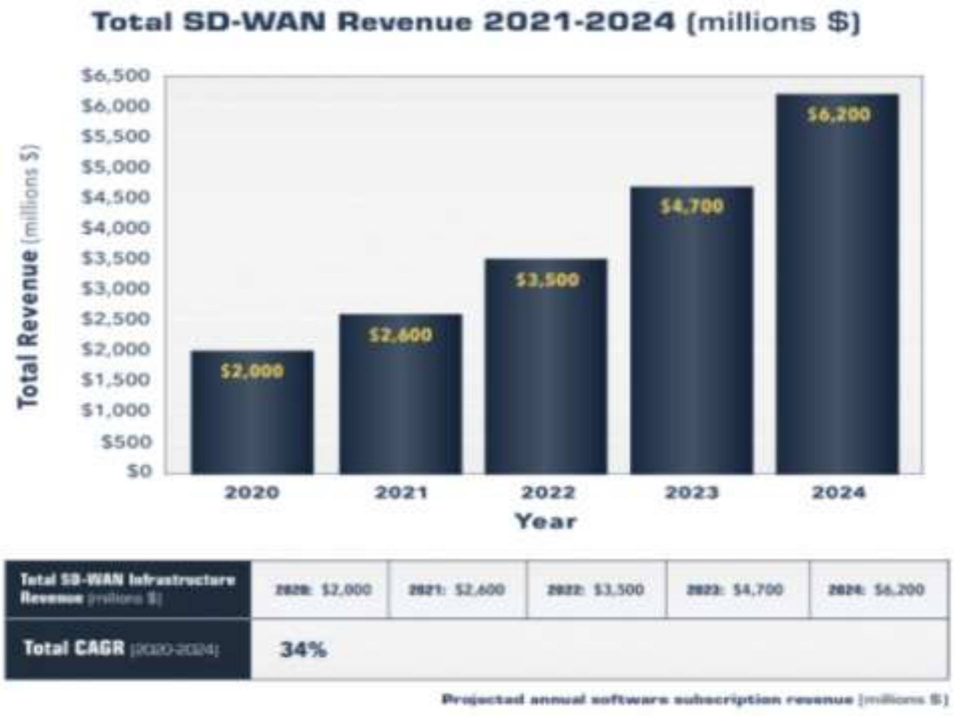


Figure 20: Key Industrial Players in the SD-WAN Markets (Greene, 2023)

In 2021, Futuriom, a research and analysis firm, evaluated the progress of SD-WAN. They highlighted the SD-WAN market's resumption of growth after being impacted by the pandemic in the first quarter of 2020. As illustrated in Figure 21, Futuriom projected a compound annual growth rate of 34% for SD-WAN software and hardware services for the years 2021, 2022, and 2023 (Raynovich, 2023).

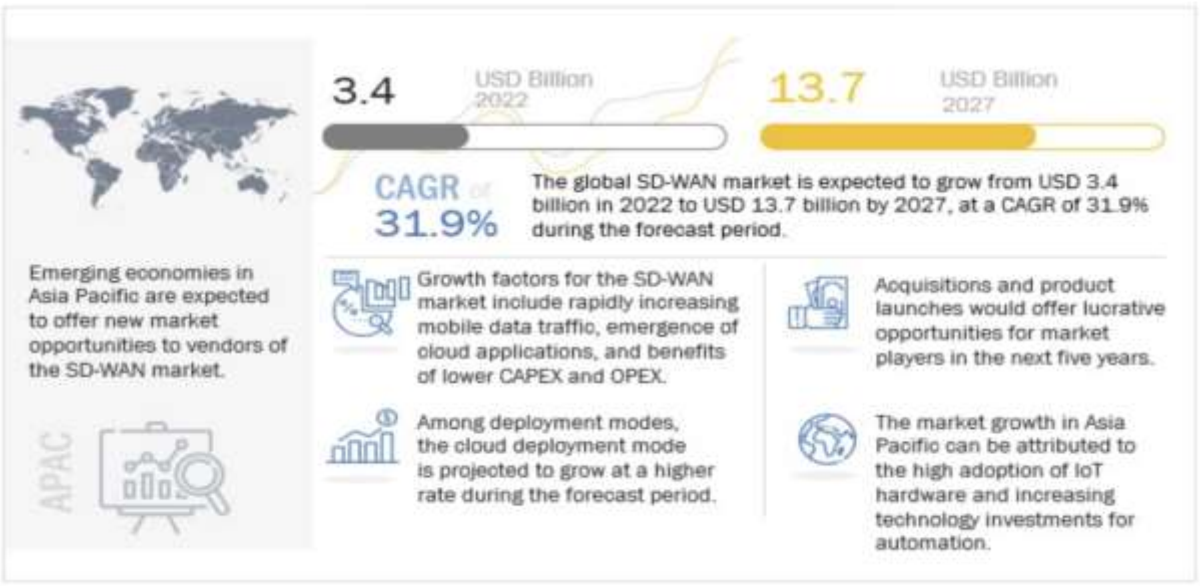


Total SD-WAN Infrastructure Revenue (millions $)	2020: $2,000	2021: $2,600	2022: $3,500	2023: $4,700	2024: $6,200
Total CAGR (2020-2024)	34%				

Projected annual software subscription revenue (millions $)

Figure 21: Total SD-WAN Estimated Revenue between 2021 to 2024 in Million Dollars (Ghode, 2023)

Upon further examination of the market report, it is anticipated that the global SD-WAN market will grow from $3.4 billion in 2021 to $13.7 billion by 2026. This translates to a compound annual growth rate of 31.9%, as illustrated in Figure 22 (Ghode, 2023).



The global SD-WAN market is expected to grow from USD 3.4 billion in 2022 to USD 13.7 billion by 2027, at a CAGR of 31.9% during the forecast period.

Emerging economies in Asia Pacific are expected to offer new market opportunities to vendors of the SD-WAN market.

Growth factors for the SD-WAN market include rapidly increasing mobile data traffic, emergence of cloud applications, and benefits of lower CAPEX and OPEX.

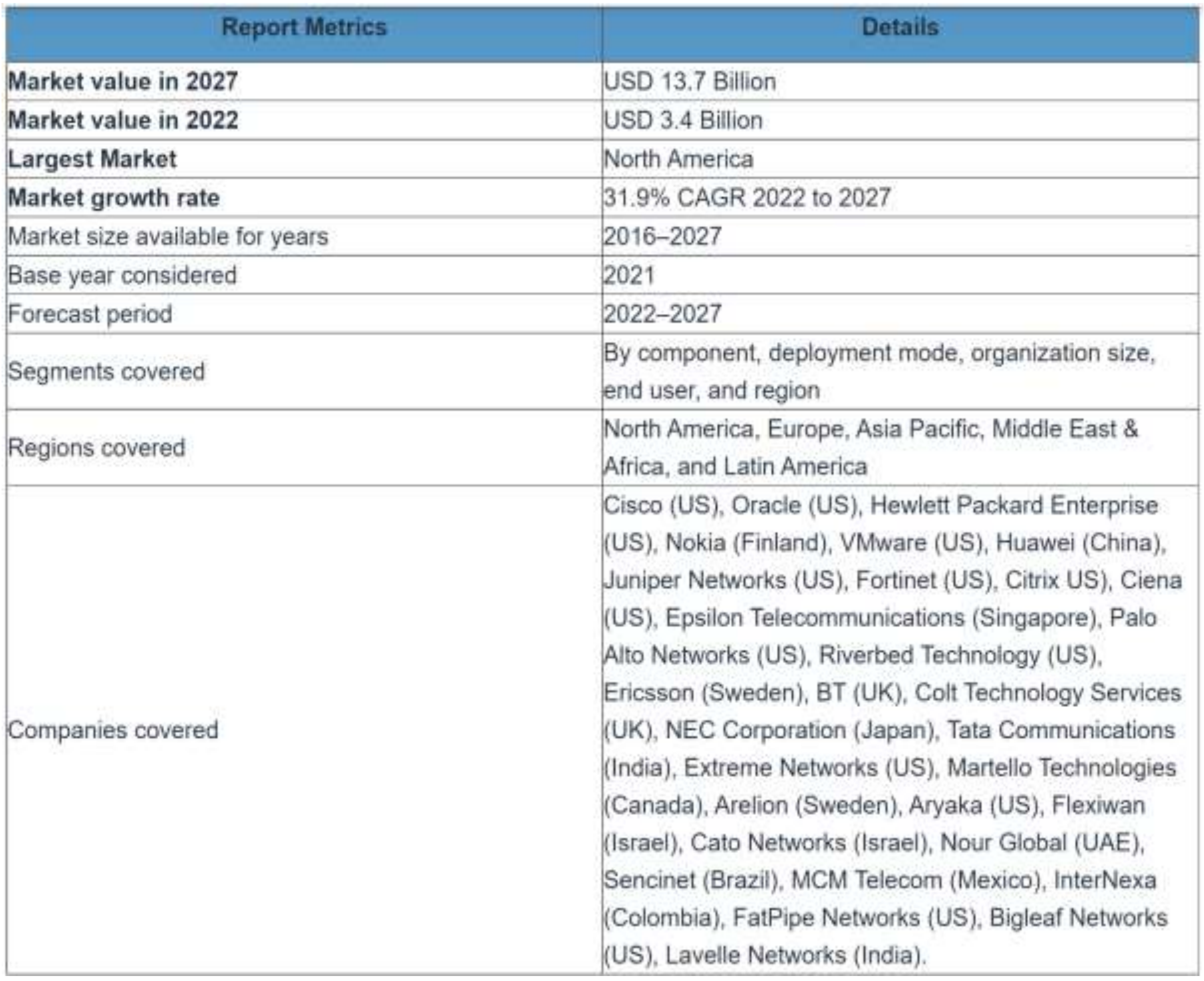
Acquisitions and product launches would offer lucrative opportunities for market players in the next five years.

Among deployment modes, the cloud deployment mode is projected to grow at a higher rate during the forecast period.

The market growth in Asia Pacific can be attributed to the high adoption of IoT hardware and increasing technology investments for automation.

Figure 22: SD-WAN Market Overview (Ghode, 2023)

Also, in terms of region, North America is the leading region in the growth and establishment of SD-WAN and some of the key metrics are summarised in Table 4.



Report Metrics	Details
Market value in 2027	USD 13.7 Billion
Market value in 2022	USD 3.4 Billion
Largest Market	North America
Market growth rate	31.9% CAGR 2022 to 2027
Market size available for years	2016–2027
Base year considered	2021
Forecast period	2022–2027
Segments covered	By component, deployment mode, organization size, end user, and region
Regions covered	North America, Europe, Asia Pacific, Middle East & Africa, and Latin America
Companies covered	Cisco (US), Oracle (US), Hewlett Packard Enterprise (US), Nokia (Finland), VMware (US), Huawei (China), Juniper Networks (US), Fortinet (US), Citrix US), Ciena (US), Epsilon Telecommunications (Singapore), Palo Alto Networks (US), Riverbed Technology (US), Ericsson (Sweden), BT (UK), Colt Technology Services (UK), NEC Corporation (Japan), Tata Communications (India), Extreme Networks (US), Martello Technologies (Canada), Arelion (Sweden), Aryaka (US), Flexiwan (Israel), Cato Networks (Israel), Nour Global (UAE), Sencinet (Brazil), MCM Telecom (Mexico), InterNexa (Colombia), FatPipe Networks (US), Bigleaf Networks (US), Lavelle Networks (India).

Table 4: Key Summary of the Market Trend of SD-WAN (Ghode, 2023)

The industrial deployment of SD-WAN has led to the availability of multiple SD-WAN solutions in the market, providing diverse client requirements, network architectures, and operational models. The various types of SD-WAN solutions include Premises-Based, Cloud-Based, MPLS-Based, Internet-Based, and Virtual-Based. Premises-Based solutions are situated at the customer's premises, with configurations and policies managed via a web application interface. In contrast, Cloud-Based solutions involve the integration of cloud virtual appliances with central WAN management software to enable the control and management of all applications, as described by (Greene, 2023). Also, MPLS-Based solutions require the placement of multiple SD-WAN appliances at network endpoints, whereas Internet-Based solutions utilise the connectivity of public internet services. Furthermore, Virtual-Based solutions are tailored for large enterprises to provide WAN optimisation, such as Microsoft Azure, AWS (AWS – Amazon Web Services), and Google Compute. Table 4 presents a comparative analysis of various vendors regarding cloud extension, scalability, load balancing, WAN optimisation, and other solutions offered to customers.

Products (source)	WAN Optimisation	Scalability	Solutions	Cloud extension	Load balancing	Gaps Identified
Adaptive-Networks SD-WAN (Parent, 2023)	Supports WAN Optimisation	Flexibility to diversify bandwidth access and provision for addition of new sites	Reduce complexity, mission-critical reliability, real-time performance, and faster bandwidth.	Microsoft Azure, AWS and Google cloud with SaaS applications	ELFIQ is the methodology used in intelligently directing inbound and outbound traffic in the network.	No traffic monitor function to monitor traffic flowing from end to end.
Aryaka SDWAN (Kiran, 2023)	Support WAN Optimisation	Allow enterprises to seamlessly connects to multiple resources in the networks	Actively involved in SmartConnect, Smart-Manage, Smart-Optimise, Smart-Cloud, Smart-Secure and Smart-Insights	SaaS as the leading applications, Azure, AWS, Google cloud and Oracle	SMARTlink allows the use of two ISP links for active configuration and balancing of traffic in the network	No traffic monitor function Integrated in the model
Barracuda SD-WAN (Barracuda, 2023)	WAN optimisation is included	Provisions for cloud clustering and auto scaling of network devices	Provide high advanced security, reduced complexity, network agility and compliance	AWS, Azure, Google cloud platform and SaaS solutions	The Barracuda Load Balancer ADC is built into the Global Server Load Balancing for high resilience and availability.	Real time traffic monitoring Solutions cannot be found.
Cato-Network SD-WAN (Norling, 2023)	Optimisation included with throughput maximisation and SLA-backed network	Provision for multiple connections amongst different offices and technologies.	Optimised global connectivity, secure internet access with a cloud and mobile support integrated	AWS, Azure, Google cloud platform and SaaS solutions	The last-mile and middle-mile optimization is effectively used for link aggregation, resiliency, latency mitigation and throughput maximisation	Real time traffic monitoring Solutions cannot be found.
Cisco Viptela SD-WAN (Franjić, 2023)	WAN optimisation is included	Viptela scalability is coined in three ways functional hardware redundancy, robust network design and the use of Overlay Management Protocol for rapid recovery	Offers visibility and analytics framework, unique capabilities, multi-cloud choice and control	Microsoft Azure, AWS and Google public cloud with SaaS applications	Load balancing mechanism includes weighted unequal per flow method, application aware routing and per application traffic pinning mechanism.	Real time traffic monitoring Solutions cannot be found.
Citrix SDWAN	Support WAN Optimisation	Cost effective, flexible and high connectivity	Provides high level of visibility, high resiliency, performance	Microsoft Azure, AWS and Google	NetScaler virtual WAN measures transit time, jitter	Real time traffic monitoring Solutions cannot be found.

(Pollock, 2023)		between branch or WAN connections.	flexible network security and virtualized applications.	cloud with SaaS applications.	and packet loss and creates a map for high performance and traffic segregation.	
Cloud-Genix SD-WAN (Lee, 2023)	Integrates with existing WAN optimisation	High possibility of integrating into existing management algorithms and creating custom database	Provides central controller, flow forwarders, ION Fabric and application fingering	Effectively uses AWS and Azure	Cloud-Genix uses path selection in ensuring high bandwidth and to meet SLA requirements.	No Google cloud with SaaS applications present and no traffic monitoring process
Cradle Point SD-WAN (Ebuah, 2023)	Available for Management streams over LTE only	Scalability is certain with the NetCloud based architecture built in the SD-WAN	The solution is integrated into cloud management and NetCloud manager	Net-Cloud, Virtual routers that support AWS	Link bonding is effectively used in ensuring high service delivery and bandwidth utilisation.	No Google cloud with SaaS applications present and no traffic monitoring process
FatPipe's SD-WAN (FatPipe-Networks, 2023)	WAN optimisation is supported	Scalable with WAN optimization and cloud-based support.	Multiple connections, policy-based management, centralised control and service chaining	Effectively uses AWS and Azure	FatPipe's uses Site Load Balancing in dynamically mapping application traffic	No Google cloud with SaaS applications present and no traffic monitoring process
Fortinet SDWAN (Laliberte, 2023)	Support WAN optimisation	Fortinet uses the FortiGate scalable and high-performance device for protection against attacks and data breach.	SD-WAN orchestration, Forti-Manager, Forti-Analyzer Cloud and management visibility plus operations	Microsoft Azure and AWS with SaaS applications	Fortinet uses the global server load balance as a DNS-based solution in easing downtime.	No Google cloud with SaaS applications present and no traffic monitoring process

FlexiWAN SDWAN (Flexiwan, 2023)	Support WAN optimisation	It is scalable and provides for multiple connections.	Solution comprises of FlexiEdge, Management Plane, Northbound API and Virtualization environment	Microsoft Azure and AWS with SaaS applications	Not specific with the procedure	Load balancing not effectively implemented
Juniper SDWAN (Khoury, 2023)	Support WAN optimisation	High scalable network design and connection	Increased visibility, application performance, lowered operational expenditure and efficient network design with SaaS-based applications	Microsoft Azure, AWS and Juniper cloud with SaaS applications	Rate limiters are used for traffic shaping and efficient bandwidth utilisation	No Google cloud with SaaS applications present and no traffic monitoring process
Nuage Networks (Nuage-Networks, 2023)	Third party WAN optimisation	Provision for new addition into the network infrastructure or architecture of the SD-WAN	Solution comprises of public cloud gateway, application aware routing and connecting disjoint underlay network	Microsoft Azure, AWS and Google cloud with SaaS applications	Application aware routing for optimally selecting the best service for uplink and downlink resources	no traffic monitoring process
Oracle SD-WAN (Cainkar, 2023)	Integrated WAN optimisation	Support scalability of the network architecture such as VMware, vSphere, Linux Kernel Virtual machine and Microsoft Hyper-V.	Oracle presents the Fail-safe SD-WAN which offer secure cloud access, increased application QoE and change WAN economics with a hybrid WAN.	Microsoft Azure, AWS and Oracle cloud applications	Not specific with the procedure	Load balancing not effectively implemented
Riverbed SD-WAN (Dante, 2023)	Support WAN optimisation	Provision for expand connectivity options using cloud platform	Offer expand connectivity options, boost operational agility, streamline branch infrastructure and accelerate site provisioning	Microsoft Azure and AWS	Offer dynamic path selection to steer network traffic onto the WAN links.	No Google cloud with SaaS applications present and no traffic monitoring process

Silver Peak SD-WAN (View, 2023)	Offers optional WAN optimisation support	Support scalability in areas of throughput, flow capacity, secure acceleration and WAN acceleration techniques.	Multi-Cloud networking, Cloud application performance and WAN optimization refresh are all provided	Microsoft Azure, AWS and Oracle cloud infrastructure	Dynamic path control integrated into the Silver Peak Unity Orchestrator is used for granular traffic control across multiple WAN links	No Google cloud with SaaS applications present and no traffic monitoring process
VMware VeloCloud SDWAN (Umbe, 2023)	Supports WAN and TCP optimisation	Supports scalability for large enterprise networks	Provisioning for fast remote branches, deploy remote and pop sites in minutes, delivering the highest quality of care	Microsoft Azure and AWS	Dynamic multi-path optimization (DMPO) methodology used for application steering, continuous monitoring and on-demand remediation	No Google cloud with SaaS applications present
Versa Secure SDWAN (Versa-Network, 2023a)	Service chaining and third-party WAN optimisation support	Supports scalability for large enterprise networks	Increased reliability and network agility, simplified branch office connectivity and optimised application performance	Microsoft Azure, AWS and Google cloud with SaaS applications	Rich set of layers 4 load balancing advanced capabilities based on source and destination IP addresses and ports recorded.	Issues with security architecture
WatchGuard SDWAN (Abraham, 2023)	Support WAN optimisation	Scalable with high level of cloud resources	Control network traffic, centralised management, monitor network traffic and manage security services	Microsoft Azure, AWS and WatchGuard cloud	use traffic management action for controlling traffic based on policy and application type	No Google cloud with SaaS applications present

Table 5: Industrial comparison of the services of SD-WAN in terms of load balancing, optimisation and e.t.c

Table 5: provides an industrial concept of SD-WAN, which can be summarised into three key aspects. First, there is a focus on WAN optimisation and scalability, with key players such as Adaptive-Networks SD-WAN, Aryaka SD-WAN, Barracuda SD-WAN, Cato-Network SD-WAN, and Cisco Viptela SD-WAN. These solutions are compatible with cloud platforms like AWS, Microsoft Azure, Google Cloud, and SaaS applications. The aim is to ensure real-time response and network performance. Secondly, there is an emphasis on cloud extension and load balancing functionalities, with solutions provided by Citrix SD-WAN, Cloud-Genix SD-WAN, FatPipe's SD-WAN, and others. These solutions enable seamless connectivity between branch offices and resources, offering network security, high application performance, and visibility. They can be integrated into various cloud applications. Furthermore, there is a focus on flexibility and scalability, primarily offered by leading SD-WAN providers such as Fortinet SD-WAN, Juniper SD-WAN, and VMware VeloCloud. These solutions provide to large clients or enterprise networks and prioritise operational efficiency, security, and agility. They incorporate advanced features like orchestration, visibility, analytics, and multi-cloud support into their architecture. Overall, these SD-WAN solutions address different aspects of network optimisation, cloud connectivity, and scalability to meet the diverse needs of organisations in various industries but there is no single solution that can meet all the current SD-WAN challenges. Thus, the proposal of a new logical SD-WAN architecture that can improve the dynamic load balancing in real time applications by implementing network traffic monitors to manage the end-to-end traffic in the architecture.

3.4: Related Work (Academic Perspective)

Over the years, the research area of SDN and SD-WAN has gained significant attention from researchers and industries alike. SDN originated in 2009 as a pet project at Stanford University, and since then, numerous innovations have emerged, including network visualisation, network function virtualisation, and SD-WAN (Cooney, 2023b). The concept of SDN and SD-WAN has expanded to various networking areas, such as reliability, scalability, load balancing, traffic management, resource allocation, and effective controller placement within the network architecture(Lerner, 2023). Some of the contributions from various researchers are summarised and areas of improvement are recognised.

(Grgurevic et al., 2021) focused on the analysis of MPLS and SD-WAN networks using GNS3 to evaluate their performance. The implemented SD-WAN comprises Cisco c3725 routers and OpenvSwitch, connecting them through separate networks using RIP routing protocol and a Mininet controller. The testbed results demonstrated that optimising the models and effectively predicting network traffic with SD-WAN can significantly enhance the overall quality of service in each network. Furthermore, (Mena Diaz et al., 2022) conducted an analysis of the advantages of SD-WAN as a replacement for traditional WAN connectivity. This analysis was carried out using the waterfall methodology with the goal of creating a virtual environment to simulate both SD-WAN and traditional infrastructures. The results indicate that SD-WAN significantly enhances infrastructure, providing increased agility and improved communication capabilities. Also, (Navarro et al., 2023) is currently doing some research work on the challenges and prospects in SD-WAN. Factors such as cost-effectiveness, improved application performance, and increased bandwidth are examined. The challenges include the lack of open SD-WAN solutions, interoperability issues with legacy networks, and the absence of adaptable real-time monitoring tools. To tackle these problems, a Multi-Agent Reinforcement Learning (MARL) framework to distribute decision-making and information exchange among agents in different WANs is proposed.

(Septyanto & Ikasari, 2021) describes the data traffic performance evaluation of an SD-WAN network using tableau software. This research involves a comparative analysis of three internet service providers (CBN, Orion, and Telkom) in terms of their performance within an SD-WAN network. The findings provide valuable insights for the company's information management practices and technology decisions, aiming to improve its overall performance. Also, it is the report of IBM which discuss on five technical challenges in managing SD-WAN which includes avoiding the temptation to over-provision, managing the change from static to dynamic connectivity, reducing network monitoring inefficiency, achieving operational agility goals, and gauging service reliability and quality with policy adherence (IBM, 2023). Moreover, (Blidborg, 2022) focused on the challenges on the implementation of monitoring systems in an SD-WAN architecture and the results obtained indicates the need for a thorough monitoring system in the edge sites is required for visibility of the underlying networks. Also, this shows there are no common standards that can be used in monitoring and network management systems yet in SD-WAN.

In the various submissions of (IBM, 2023; Navarro et al., 2023) and (Blidborg, 2022) it clearly points out the limitations and the need for a new architecture or procedures in the monitoring process of SD-WAN architecture. This research has identified the need to improve and make changes to the monitoring process of the current SD-WAN architecture. Also, further research by (Liu & Li, 2015) investigated the scalability of SD-WAN and identified issues with TCAM (Ternary Content-Addressable Memory) in the data plane and bottlenecks in the control plane due to an ineffective offload mechanism and physical distribution. To address these problems, (Liu & Li, 2015) employed partitioning schemes and data plane mechanisms such as MPLS-based and Label switching for offloading to the core network. They used schemes such as DIFANE, HyperFlow, DevoFlow, Kandoo, and ONIX for network partitioning and filtering information before sending it to the controllers and operating system.

(Duliński et al., 2020) proposed Dynamic Traffic Management (DTM) for SD-WAN inter-cloud communication, which dynamically reacts to traffic changes on inter-domain links, shifting most of the traffic management in the network to another domain link. They used R-vector prediction and compensation vector calculation procedures for optimal traffic management. Meanwhile, (Kim & Feamster, 2013) developed Procera, an event-driven network control platform based on SDN, to address issues related to network management such as frequent changes to network conditions and state, better visibility/control tasks in network diagnosis and troubleshooting, and difficulties in high-level languages network configuration. However, Procera is limited to only four control domains, which might not be a perfect solution for scalability in SDN.

(Hu et al., 2021) present Trident, a lightweight and efficient software network monitoring device that is policy-based and programmable. Trident is designed for packet capturing, filtering, and delivery, and utilises a packet classification algorithm and a cached flow form for optimisation. According to the evaluation and demonstration, Trident has high efficiency with minimal interference when implemented in SD-WAN. Also, (Tootaghaj et al., 2020) introduce Homa, an efficient topology and route management approach for SD-WAN overlays. Homa uses informed knowledge of the network state to make decisions in the SD-WAN network, reducing network reconfiguration costs during updates. The method employs greedy algorithms to solve the Min-Cost problem, and experimental results on a real network topology demonstrate optimal performance in terms of disrupted flows and disruption cost.

In (Shu et al., 2016), Traffic Engineering on SDN is presented, which focuses on the issue of network measurement and management. The research considers various measurement methodologies (ProgMe, ISTAMP and generic) and management approaches to address the issue of network measurement. Similarly, (Troia et al., 2020) proposed deep reinforcement learning algorithms to improve service availability and network restoration and security in SD-WAN. The study reviewed baseline traffic engineering algorithms and proposed three types of deep learning reinforcement algorithms (TD-λ, Policy gradient and deep Q-learning) that show overall improvement in network performance.

(Truong et al., 2016) propose using Route Flow and VROOM to improve network management with SDN. They present an updated version of Route Flow called Flow Map, which allows IP routing and virtualisation in OpenFlow networks. Flow Map allows the operation of virtual topologies in the controller and dynamically maps the underlying physical infrastructure through the programmable interface. The study reports temporary inconsistencies causing packet loss and suggests live migration as a solution for simple management-driven tasks.

(Agarwal et al., 2018) analysed revenue generation for service providers using a priority traffic engineering framework in SDN. They used Mininet's application software, POC controller and open switch edge devices to analyse quality-of-service constraints like delay, bandwidth, throughput, and latency. The results showed improved link utilisation, reduced packet drops and latency. On the other hand, (Tuncer et al., 2015) proposed a new SDN-based adaptive resource management and control framework for fixed backbone networks, which was implemented in three layers, and the performance was analysed using heuristic. Future work is proposed on the unequal cost of splitting in OpenFlow and cache management.

The research conducted by the three authors ((Truong et al., 2016); (Agarwal et al., 2018) and (Tuncer et al., 2015)) holds significance for SD-WAN as they introduces opportunities to enhance network management capabilities, enable programmable features, and increase flexibility in the architecture. Additionally, their work addresses temporary inconsistencies within the architecture and proposes programmability features specifically designed for SD-WAN. These advancements are achieved through the introduction of techniques such as Route Flow, adaptive resource management, and flow mapping, all aimed at improving network efficiency. But these methods are not all encompassing as there are gaps such as scalability, real time implementation and cost of the overall network architecture.

In (Mora-Huiracocha et al., 2019), an SD-WAN is implemented to interconnect two software-defined data centres, with the use of an SD-WAN controller to monitor QoS and prioritise traffic. The proposed methodology is evaluated using two virtualised servers via VMware, and the results show improved QoS and policy aggregation. In a similar application, (Wang et al., 2018) presents the integration of an intelligent rule management scheme (IRMS) for SDN, where flow rules are divided into path-based and node-based rules for optimal performance. The simulation results show improved performance in terms of resource-saving, flexibility, and QoS.

Furthermore, (Aydeger et al., 2015) investigated the resilience of SDN for smart grid communication, enabling a backup wireless interface connection in the OpenDaylight SDN controller for failures in the wired connection. Mininet and NS-3 are integrated using OpenFlow protocol to observe the traffic performance characteristics. In (Troia, Mazzara, Zorello, & Pattavina, 2021) an SD-WAN Traffic Engineering application was developed for resiliency in SD-WAN with eBPF (extended Berkeley Packet Filter) monitoring, improving, and managing traffic at the edge of the municipal network. The results showed a significant increase in network performance and service availability, ensuring high network resiliency in the case of transmission disruption and link failures.

In similar research, (Yang Zhang et al., 2021) researched improving SD-WAN resilience by using WAN-aware multi-path TCP to aggregate multiple WAN links into a "big pipe." This approach reduced the impact of path failures, enabled faster recovery, and enhanced WAN link utilisation while supporting application-aware SD-WAN traffic engineering. In a testbed and real-world deployment, WAN-aware multi-path TCP showed better performance gains than the existing SD-WAN solutions. (Montazerolghaem et al., 2017) focused on improving the performance of Session Initiation Protocol (SIP) using Software-Defined Networking (SDN) and Network Function Virtualisation (NFV). They achieved this by decoupling SIP proxy functionalities from the infrastructure layer to a programmable software controller, resulting in lower response times, high throughput, resource efficiency, cost reduction, load balancing, unified management, and easy programmability.

Furthermore, (Fajjari et al., 2017) proposed an SDN scheme for quality-of-service path allocation in WAN, which creates a computational problem based on an integer linear program and is resolved using the branch-and-cut algorithm. Numerical results show better network resource consumption performance and satisfaction compared to other algorithms. In addition, (Hou et al., 2019) developed a multi-controller deployment algorithm in a hierarchical architecture for SD-WAN to address limited processing power in WAN. This approach involves an improved Louvain algorithm in partitioning the data plane, deploying the root and domain controller for reliable and effective load balancing technique. The hierarchical architecture divides the control plane into multiple levels, and simulation results showed improved performance for delay, better load balancing technique, and reliable control plane architecture.

(Heller et al., 2012) discussed the controller placement problem in SDN, focusing on the number of controllers located in each SDN network and their location in the network architecture. They used internet2 OS3E to simulate the impact of these indicators in real network environments and suggested placing controllers at the starting point to reduce latency and possibly using a dynamic approach to balance switches among controllers in the network. Similarly, (Amiri et al., 2021) formulated an optimised controller placement for SD-WAN using the controller placement problem as a Facility Location Problem to achieve an average propagation latency and load balancing among controllers. The solution offers the optimal number and placement of controllers to achieve the average packet propagation and controller processing latency in the network, and simulation results indicate an adequate threshold for average message arrival rate to achieve optimal load balancing among controllers in the SD-WAN network.

In a related work, (Cai et al., 2019) defines an effective load balanced controller placement to reduce the average delay between switches and controllers in SD-WAN. Cai et al. (2019) formulated the problem as a K-median problem and implemented a three-phase approach to determine controller locations, assign switches to controllers, and update controller placement. The results show a reduction in average delay and numbers of controllers to avoid overload in the SD-WAN architecture. Similarly, (Wang et al., 2017) used a clustering-based network partition algorithm (CNPA) and multi-controller placement algorithm to reduce end-to-end and queuing latency between switches and controllers using MATLAB (Internet Topology Zoo). Both studies demonstrated improvements in latency and load balancing through their proposed approaches.

(Rajagopalan, 2020) discusses various studies on load balancing in SD-WAN and explains the importance of load balancing for effective management of traffic across various ISPs to optimise routing, reduce bottlenecks and prioritise applications. Similarly, (K. Yang et al., 2019) proposes a low-complexity algorithm, SAKPM (Simulated Annealing Partition-based K-means), with two cost functions to measure controller suitability and performance metrics such as processing capacity, network reliability, and propagation delay, which show an improvement in load balancing and performance metrics.

The study by (Yuniarto & Sari, 2021) evaluates multipath deployment in SD-WAN to improve throughput and analyses packet loss and delay from branch to headquarters. The implementation uses the Ryu controller and data plane devices designed in Mininet, with throughput results showing 20Mbps in the Multipath deployment compared to traditional networking with a single link of throughput. In (Cimorelli, 2018) worked on the SDN workload balancing and QoE (Quality of Experience) control in next generation network infrastructure. Two resource management frameworks were designed, utilising a centralised logically/physically distributed SDN control plane and load balancing algorithms based on gaming theory, to balance SDN control traffic and user quality of experience at the network edge. The proposed algorithm is called Wardrop equilibrium and was implemented using the Mininet and MATLAB tools. The future research will focus on improving convergence properties of the algorithm and multi-agent coordination for large scale deployment.

Likewise, (Guo et al., 2021) propose a scalable routing approach called Hybrid-Flow to achieve load balancing in SD-WANs with low processing overhead by identifying crucial flows and rerouting on the forwarding path. They propose a heuristic algorithm to solve the problem based on the given resources, and simulation results show that the proposed approach maintains steady network performance with little control overhead. (Al-Sadi, 2018) developed a placement algorithm for the controllers of virtual networks (COVN) in SD-WANs with multiple VNs. The algorithm optimises latency, reliability, optimal resource utilisation, load balancing, and recovery of controller failure. The COVN algorithm is used to calculate the controller placements of the physical controllers for every virtual SD-WAN independently. Implementation was done using the Mininet Emulator, and the results show better reliability and decreased latency.

In a related work, (Shozi et al., 2019) proposed a methodology using the Agglomerative Hierarchical Clustering algorithm (AHC) to solve the issue of switch placement problem and SD-WAN partitioning plus clustering validity approach to decide the number of clusters. Their algorithms were implemented using MATLAB and the simulation results show decreased end-to-end delay latency with the effective application of the clustering-based network partition algorithm (CNPA) and multi-controller placement algorithm. (Sahoo et al., 2019) proposed improving end-users' utility in SD-WAN systems by considering the requirement of layer-3 and layer-2 controllers in WAN and LAN separately. They proposed a load prediction approach to reduce the workload of the controller, and simulation results show improvements in prediction error, Quality of Service, and end-users' utility in a heavy traffic scenario. The article discusses a study by (Qiu et al., 2017) that introduces the concept of ParaCon, which focuses on the parallel control plane for scaling up path computation in SDN. They proposed an asynchronous path computation algorithm for the modified version of POX controller using Mininet with python scripts. The implementation results show reduced transmission overhead, improved performance, and better convergence time compared to existing legacy networks.

The study by (Lin et al., 2016) proposes a nonlinear optimization framework to control traffic forwarding paths for different switches in SDN to reduce average traffic delay. The polynomial time approximation algorithm (PTAA) enables fast convergence among switches using the alternating direction method of multipliers (ADMM). The evaluation shows that the proposed algorithm reduces delay by 80%. The second study by (Manel & Habib, 2017) proposes a multi-hop clustering algorithm for an efficient MPLS-based source routing scheme in SD-WAN to solve problems such as controller-to-switch communication overhead, switch-to-switch communication overhead, and flow table consumption. The evaluation shows improvements in bandwidth overhead compared to MPLS-based forwarding.

The research by (Golani et al., 2018) proposed a fault-tolerant smart routing system that uses SD-WAN to monitor a WAN in real-time. The system provides high availability, reliability, and low latency, while also reducing packet loss. The algorithm used in the system is resilient to link failures and was tested in a real-world environment using Open-Lab at Juniper networks. The system achieved good performance metrics such as hop count, latency, and link reliability.

Also, (Patel et al., 2016) implemented a combination of NFV and SDN in OpenStack cloud for network security and improvement, using orchestration to manage the SDN controller and the data plane connected to the cloud network to handle network routing and traffic forwarding. Performance metrics such as latency and throughput showed improved functionality and security compared to traditional algorithms. Meanwhile, (Ahmed & Ramalakshmi, 2018) evaluated the performance of centralised and distributed SDN controllers for load balancing applications. The study involved the implementation of load balancing algorithms in a POX controller integrated with Mininet software for analysis. It was observed that distributed controllers work faster than centralised controllers, and Weighted round-robin transacts more in both architectures than the random algorithm.

Furthermore (Troia et al., 2020) discusses the implementation of an open-source SD-WAN enterprise networking service that changes forwarding rules based on service requirements and data monitored. The study uses a testbed to examine delay and packet loss in a network scenario. Also, (Troia, Mazzara, Zorello, & Maier, 2021), evaluates the performance of overlay network services for delay-sensitive applications such as video and voice over IP streaming, with a focus on resilience and fast recovery in case of network failure. The study integrates extended Berkeley Packet Filter (eBPF) and SD-WAN traffic engineering with a monitoring system and shows improved network performance, but with limitations in terms of scalability.

Also, (Carvajal et al., 2021) implements and demonstrates a corporate transformation of SD-WAN using a case study of the Cosentino Group, which results in better user experience with SaaS models. The study also identifies challenges such as contract management aligning with operational and capital expenditure. Furthermore, (Scarpitta et al., 2021) describes an open-source SD-WAN solution called EveryWAN, implemented using the Mininet emulation environment. The solution allows for testing and measurement of control plane parameters such as scalability, time, and throughput for different WAN topologies, NATed environments, and broadband technologies.

In similar research, (Zakurdaev et al., 2022) proposes a dynamic tunnelling splitting approach based on demand to reduce delay and load in the corporate network for cloud-based applications. The architecture is implemented using Mininet, and results show a reduction in delay, effective policy management, and preservation of the benefits of split tunnels. (Guo et al., 2022), proposes a solution to controller failures by controlling resiliency and flow programmability features. The study introduces a retroFlow+ mechanism that reduces overhead from offline switches to controllers and addresses limitations such as not updating route flow paths, controller overload, and network performance degradation. Also, (Amiri et al., 2021) focuses on the optimal placement of controllers for SD-WAN, aiming to examine network parameters such as delay, load balancing, energy saving, and reliability. The study models the controller placement problem as a Facility Location Problem, examining metrics such as average message arrival time, propagation latency, and controller processing latency. Results show improved load balancing between controllers by setting appropriate thresholds and an increase in the number of controllers with lower propagation latency.

Furthermore, (Hamdan et al., 2022) presents the Bandwidth and Congestion Aware routing (BACAR) for hybrid networks using SDN, which identifies congested links and degradation events through active delay measurement. The implementation in Mininet shows enhanced flow performance and improved network resource utilisation by identifying bandwidth fluctuations in long-distance wireless links and searching for alternative routes for sensitive flows. The study, (Yi Zhang et al., 2021) focuses on an improved sharing algorithm that implements a distributed data backup and recovery (DDBR) method for SD-WAN architecture, dividing data files into various segments and storing them in switches. The offline-online dual backup framework is proposed to reduce communication overhead and data storage, with encryption and confidentiality added for rapid data recovery. Furthermore, (Manova et al., 2022) conducted a comparative analysis of the quality of service and performance of MPLS, EoIP, and SD-WAN. The findings indicated that MPLS outperformed EoIP and SD-WAN in terms of performance and quality of service. However, it was noted that this superiority of MPLS comes at a higher cost, particularly as enterprises and cloud computing continue to expand. It is worth mentioning that MPLS can still be incorporated into the SD-WAN architecture.

(Guardabaxo & Pavani, 2022) proposes a method for prioritising HTTP/2 packets in SD-WAN to improve Quality of Experience for web users. The approach is demonstrated using a prototype model built with a Ryu controller, Netty-based HTTP/2 server, OpenVSwitch and Mininet, which shows significant improvement in Quality of Experience for the HTTP/2 protocol. (Lemeshko et al., 2021) focuses on improving traffic engineering fault-tolerant routing in SD-WAN using flow-based mathematical models. The approach aids in load balancing at data planes and gateway protection scheme for SD-WAN, and the result confirms the effectiveness of the load balancing algorithm. Also, (Yujie et al., 2020) discusses a new algorithm implemented in SDN for controller placement that aims to reduce network propagation delay and improve network performance. The algorithm uses clustering and information entropy to automatically determine the number and position of controllers in the network, and simulations show that it outperforms other algorithms in terms of propagation delay. This involves using traffic engineering to configure and update network behaviour by changing routing conditions, and a set of algorithms with provable performance and regret analysis is presented. Empirical testing on two topologies shows a reduction in the total cost over the time horizon. The article by (Šeremet & Čaušević, 2019) presents a comparative study of policy-based routing on WAN links in classical IP/MPLS and an application of SDN controller to manage traffic on wireless area network links. The research enables application analysis in site automation/deployments for operations and configuration with improved performance. Also, (Sainz et al., 2020) focuses on deep packet inspection for intelligent intrusion detection in software-defined industrial networks using machine learning and deep learning algorithms, and the results point towards the validity of the intrusion detection system in the implementation of SDN.

Table 6.0: provides a summary of academic and industrial perspectives from various researchers, outlining their research focus on topics such as load balancing, traffic management, scalability, resilience, data centres, cloud extensions, controller placement, WAN optimisation, and Quality of Service, as well as the solutions they offer. Nevertheless, in the context of this literature review, there exists a lack of comprehensive information regarding the fusion of a traffic monitor functions and the SD-WAN controller. In reference to Table 6 each of the research papers are indicated as "Yes" and "No" to denote whether specific research has been addressed on enhancing the operational processes of SD-WAN. Through the review of the academic and industry references outlined in Table 6, it becomes clear that current SD-WAN controllers do not integrate a distinct traffic monitoring function responsible for regulating traffic management policies that can send updates to the SD-WAN controller to facilitate dynamic load balancing. Hence, the primary aim of this study is to create an innovative SD-WAN framework integrating a traffic monitoring feature designed to enable dynamic load balancing across end-to-end protocol operations. The traffic monitor is responsible for capturing and overseeing network traffic, and subsequently transmitting route updates to the controller to facilitate load balancing among diverse WANs.

Attributes -> Author	Controller Placement and Data Centres	Traffic Management and Engineering	Scalability and Resilience	WAN Optimization and Quality of Service	Load balancing	Solutions (Quality of experience and services)	Cloud Extensions	Congestion control and intelligent
Agarwal et al., 2018	NO	YES	NO	NO	NO	NO	NO	NO
Ahmed & Ramalakshm, 2018	NO	NO	NO	NO	YES	NO	NO	NO
Al-sadi, 2018	Yes	NO	NO	NO	NO	NO	NO	NO
Amiri et al., 2021	YES	NO	NO	YES	NO	NO	NO	NO
Aydeger et al., 2016	NO	NO	YES	NO	NO	NO	NO	NO
Barracuda, 2020	NO	NO	YES	YES	YES	YES	YES	NO
***Building the Future with Versa Networks*, n.d**	NO	NO	YES	YES	YES	YES	YES	NO
Carvajal et al., 2021	NO	NO	NO	YES	NO	YES	YES	NO
Cradlepoint, 2020	NO	NO	YES	YES	YES	YES	YES	NO
Cai et al., 2019	YES	NO	NO	NO	NO	NO	NO	NO
David & Dante, 2023	NO	NO	YES	YES	YES	YES	YES	NO
Doyle, 2018	NO	NO	YES	YES	YES	YES	YES	NO
Dulinski et al., 2020	NO	YES	NO	NO	NO	NO	NO	NO
E. K. Ali et al., 2018	YES	NO	NO	NO	NO	NO	NO	NO
Evers, 2023	NO	NO	YES	YES	YES	YES	YES	NO
2I	YES	NO	NO	NO	YES	NO	NO	NO
Federico Cimorelli, 2017	NO	NO	NO	NO	YES	NO	NO	NO
Frederick Parent, 2023	NO	NO	YES	YES	YES	YES	YES	NO
Fajjari et al., 2017	NO	NO	NO	YES	NO	NO	NO	NO
Guardabaxo et al., 2022	NO	NO	NO	NO	YES	YES	NO	NO

Attributes -> Author	Controller Placement and Data Centres	Traffic Management and Engineering	Scalability and Resilience	WAN Optimization and Quality of Service	Load balancing	Solutions (Quality of experience and services)	Cloud Extensions	Congestion control and intelligent
Guo et al. 2021	NO	YES	YES	YES	YES	NO	NO	NO
Guo et al. 2022	NO	NO	YES	NO	NO	NO	NO	YES
G. Wang et al., 2018	YES	NO	NO	NO	NO	NO	NO	NO
Golani et al., 2018	NO	NO	NO	YES	NO	NO	NO	NO
Hamdan et al., 2022	NO	NO	NO	NO	YES	NO	NO	YES
Heller et al., 2012	YES	NO	NO	NO	NO	NO	NO	NO
Hou et al., 2019	YES	NO	NO	NO	NO	NO	NO	NO
Khoury, 2023	NO	NO	YES	YES	YES	YES	YES	NO
Kiran, 2020	NO	NO	YES	YES	YES	YES	YES	NO
Kim & Feamster, 2013	NO	YES	NO	NO	NO	NO	NO	NO
L. Wang et al., 2018	NO	YES	NO	NO	NO	NO	NO	NO
Lemeshko et al., 2021	NO	YES	NO	NO	YES	NO	NO	NO
Liu & Li, 2015	NO	NO	YES	NO	NO	NO	NO	NO
Lin et al., 2016	NO	YES	NO	NO	NO	NO	NO	NO
Lars Norling, n.d.	NO	NO	YES	YES	YES	YES	YES	NO
Mora-Huiracocha et al., 2019	YES	NO	NO	NO	NO	NO	NO	NO
May, 2020	NO	NO	YES	YES	YES	YES	YES	NO
Patel et al., 2017	NO	NO	NO	YES	NO	NO	NO	NO
Pollock, 2020	NO	NO	YES	YES	YES	YES	YES	NO
Qiu et al., 2017	NO	NO	YES	NO	NO	NO	NO	NO
Sebastian, 2021	NO	YES	YES	YES	YES	YES	NO	NO
Sagar et.al., 2020	NO	YES	NO	YES	YES	NO	NO	NO
Sanjay & Ajit, n.d	NO	NO	YES	YES	YES	YES	YES	NO
Scarpitta et.al., 2020	NO	NO	YES	NO	NO	NO	YES	NO
Shozi et al., 2019	YES	NO	NO	NO	NO	NO	NO	NO
Shu et al., 2016	NO	YES	NO	NO	NO	NO	NO	NO

Attributes -> Author	Controller Placement and Data Centres	Traffic Management and Engineering	Scalability and Resilience	WAN Optimization and Quality of Service	Load balancing	Solutions (Quality of experience and services)	Cloud Extensions	Congestion control and intelligent
Seremet & Causevic, 2019	NO	NO	NO	YES	NO	NO	NO	NO
Sainz et al. (2020)	NO	NO	YES	YES	NO	NO	NO	NO
***SD-WAN for Hybrid Multi-Line WAN with High Security*, n.d**	NO	NO	YES	YES	YES	YES	YES	NO
Truong et al., 2016	NO	YES	NO	NO	NO	NO	NO	NO
Troia et. al., 2021	NO	YES	NO	YES	NO	NO	NO	NO
Troia et. al. (S), 2021	NO	NO	YES	YES	NO	NO	NO	NO
Troia et. al. (S), 2022	NO	NO	YES	YES	YES	NO	NO	NO
Tuncer et al., 2015	NO	YES	NO	NO	NO	NO	NO	NO
View, 2023	NO	NO	YES	YES	YES	YES	YES	NO
Yang et al., 2019	YES	NO	NO	NO	NO	NO	NO	NO
Yuniarto & Fitri Sari, 2021	NO	NO	YES	YES	YES	NO	NO	NO
Yujie et al., 2020	YES	YES	NO	NO	YES	NO	NO	NO
Zakurdaev et al., 2022	NO	NO	NO	YES	NO	NO	YES	NO
Zhang et al., 2022	YES	NO	NO	NO	YES	NO	NO	NO
Zheng et al., 2022	NO	YES	NO	NO	YES	NO	NO	NO
Zmora, n.d	NO	NO	YES	YES	YES	YES	YES	NO
Ikiomoye et.al., 2024	NO	NO	NO	YES	YES	YES	NO	YES

Table 6: Summary of Academic Literature and Industrial perspective based on various sources.

3.5: Summary

The research encompasses academic literature covering various aspects such as scalability, controller placement, load balancing, and traffic management in SDN and SD-WAN. It also presents a comparative study of industry-leading SDN and SD-WAN products, focusing on the solutions offered, cloud extension capabilities, load balancing mechanisms, and WAN optimisation policies provided to clients. Furthermore, the chapter explain the industrial perspective of SD-WAN, including the standardisation processes involved. Through the review of academic and industry sources, it becomes clear that existing SD-WAN controllers do not incorporate a traffic monitor function that governs traffic management policies and lack the ability to adapt to dynamic changes in end-to-end protocol performance when making real-time load-sharing decisions. Consequently, this research proposes an SD-WAN controller that automates the network infrastructure and intelligently controls packet flow decisions between networks based on traffic management policies and the changing behaviour of transported applications.

CHAPTER FOUR: RESEARCH METHODOLOGY

Chapter four of the research work focuses on the continuation of chapter three, which primarily aimed to identify gaps in the existing literature and understand the industrial perspective. In this chapter, the methodology, design, and measurement tools for SD-WAN technology are discussed.

4.1: Overview

The widespread adoption of SD-WAN in various industries and research communities has triggered a focused effort to develop innovative ideas for enhancing its efficiency, particularly in terms of congestion control and latency reduction. The literature review identified a gap within the current SD-WAN development and research in that controllers lack the ability to adapt to dynamic changes in end-to-end protocol performance when making real-time load-sharing decisions. Hence, this research proposes a new modified SD-WAN architecture that introduces a Traffic Monitor function, specifically designed for deep packet inspection of end-to-end traffic flows. The main objective is to dynamically adapt the Quality-of-Service (QoS) routing by facilitating information exchange between the traffic monitoring system function and the SD-WAN controller. Hence, this facilitates the dynamic analysis of end-to-end protocols, particularly focusing on the TCP control plane, to facilitate dynamic load balancing in the SD-WAN by reacting to changes in any congestion experienced by each traffic flow.

A widely accepted best practice in modern network design and implementation is the network lifecycle model introduced by Cisco known as PPDIOO (Prepare, Plan, Design, Implement, Operate, and Optimise). This model encompasses multiple stages: preparation involving strategy and requirements, planning specific network requirements, designing a detailed network design or diagram, implementing through testing and validation, operating in production, and monitoring, and optimising through adjustments and optimisation (Digg, 2023). The design implemented in this research is based on the review of the PPDIOO methodology proposed by Cisco. To achieve this, the SD-WAN architecture is prepared by thoroughly understanding the business requirements and assessing the existing infrastructure. All necessary resources are then secured for the project. Also, the various components involved in the network topology, such as PCs, switches, routers, and NAT devices, are identified and integrated into the architecture. Once the implementation is successful, the operational mode is activated to enable functions like network monitoring and management. Furthermore, the SD-WAN network infrastructure is designed by measuring key metrics and performance indicators. Figure 23: shows a diagrammatic representation of the network design life cycle model.

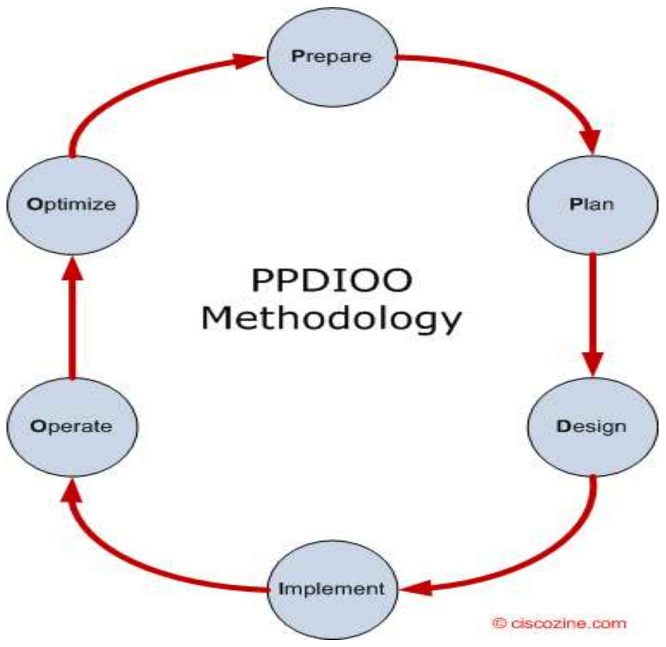


Figure 23:Network Design lifecycle model implementation (Semperboni, 2023)

4.2: Emulators (Simulation Environment)

A network emulator is a software tool that is used in testing the performance of a real network, proof of concept, quality assurance and troubleshooting which is effectively used in replicating or predicting network behaviour (Cole, 2023). Multiple emulators are developed for designing and implementing networks, but in this project GNS3 (version 2.2.20) is used. The choice of GNS3 (version 2.2.20) is selected due to its stability and compatibility, providing a real-time network simulation environment though with some limitations. GNS3 enables configuration, emulation, troubleshooting, and performance testing of both real and virtual networks, supporting various network topologies. While other emulators such as Omnet++, NS2, and OPNET serve similar purposes, GNS3 stands out as a user-friendly option with integrated Wireshark support and compatibility with docker containers. These features enable programmable functionalities and automation in SD-WAN network design and implementation. NetSim and other similar emulators such as Omnet++, NS2 are leading network emulators that enable the creation of networks for studying performance and behaviour. While it is effective for analysing the TCP window and evaluating TCP/IP with different congestion algorithms, it has limitations in implementing SD-WAN architecture and lacks user-friendly configuration options, thus the need of GNS3. Furthermore, it is Wireshark, a robust network packet analyser software that captures and verifies network packets, providing detailed information and statistics. Its integration within GNS3 and NetSim eliminates the need for additional hardware, making packet data capture more realistic and efficient for preliminary analysis (Combs, 2023).

4.3: Routing Protocols (WAN Connection)

The implementation of SD-WAN relies on utilising multiple WAN connections efficiently. In this thesis network design, three ISP or WAN's provisions are incorporated. Initially, the Routing Information Protocol (RIP) is implemented amongst the various three ISP and the edge routers in the network design to ensure the smooth transmission of packets between various nodes in the network. Static routing is employed to implement the functionality of the SD-WAN controller by updating the routing table in the edge routers. Also, the static route is implemented for the SD-WAN controller to transmit the updated routes to the edge routers, facilitating the implementation of necessary modifications.

4.4: Performance Metrics at End-to-End Analysis

This section centres on evaluating the performance metrics of the end-to-end protocol within the modified SD-WAN architecture. The performance metrics are utilised to evaluate the behaviour and efficiency of the designed network, with the goal of achieving a quality of service and enhancing overall network performance. These metrics comprehensively assess different aspects such as network latency, bandwidth utilisation, throughput, and quality of service. By analysing these metrics, valuable insights into the network's performance are gained, allowing for identification of areas that may require improvement between the end-to-end in the network architecture. The following performance metrics are considered in this research.

4.4.1: Packet Loss Percentage

To ensure network quality and reliability, it is essential to analyse the percentage of packet loss. This metric focuses on evaluating the integrity of all captured packets between end-to-end. Additionally, computing the percentage of packets that are received incorrectly or lost provides insights into potential congestion issues. These examinations help in assessing network performance and identifying possible congestion scenarios. In this thesis, the percentage of packet loss is computed in the traffic monitor function and is computed by examining the total number of packets loss with respect to the total number of packets received expressed in percentage. The packet loss is computed by comparing the packet sniff time with the expected arrival time of the packets. If the difference in time is greater than a certain threshold (1 second), the packet is considered lost. Thus, in the consideration of this research, 5% of percentage loss would likely indicate high congestion while greater than 2% will be indicating a medium congestion level.

4.4.2: Average Latency

Latency is a crucial metric in network performance evaluation, specifically measuring the time it takes for a packet to travel from its source to the destination. This metric directly impacts the efficiency and responsiveness of captured packets within the network. Analysing latency aids in identifying bottlenecks and optimising traffic flow for improved network performance at the end-to-end. In this thesis, the traffic monitor function incorporates the calculation of average latency, which is determined by the difference between the packet's sniff time and its transmission time. The monitor captures both the packet's sniff time and its relative time, enabling the measurement of latency by computing the difference between these parameters. Specifically, latency is assessed by subtracting the packet's relative time, representing its relative timing, from packet.sniff_time (this is time a packet flowing through the network is detected and observed), indicating the precise moment the packet was captured. This calculation yields the latency value for the packet under analysis. These values are recorded for each packet and the average mean is taken for all the packets captured. Based on the average mean taken for the accumulated packets a value of 0.5seconds indicates high congestion while more than 0.2seconds indicates medium congestion.

4.4.3: Congestion Level

This research focuses on evaluating the congestion level of different packets by analysing their packet loss percentage, average latency, and variations in the window size value of the TCP packets. By utilising this metric, the study aims to assess and monitor congestion levels with the objective of achieving optimal network performance. Thus, the relationship between percentage of packet loss, average latency and variations in window size are the basic parameters used to detect congestion in the traffic monitor functions.

4.4.4: Bandwidth Utilisation

Gaining an understanding of the bandwidth utilisation across various networking paths or links is crucial for effective SD-WAN management. These parameters offer valuable insights into identifying over-utilised or under-utilised links within the WAN at the end-to-end protocol. This information facilitates the balancing of traffic load and optimisation of resource allocation. This thesis proposed three categories of bandwidth, capacity bandwidth, utilised bandwidth, and outstanding bandwidth. The capacity bandwidth indicates the bandwidth of the link while the utilised bandwidth is the bandwidth used by the link and outstanding bandwidth refers to the remaining bandwidth.

4.4.5: Quality of Service (QoS)

One of the central focuses of this research is maintaining a high level of Quality of Service (QoS) to enhance both user and application experiences. This objective is achieved by implementing effective traffic policies and diligently adhering to flow rules. The research involves sending route updates when links are congested leading to sharing of the load amongst various WANs. Therefore, these performance metrics outlined previously serve as the foundation for assessing the end-to-end performance of both the current and re-modified SD-WAN architectures. These metrics are pivotal for gauging and quantifying improvement resulting from the adjusted SD-WAN structure. A thorough analysis of these parameters is expected to provide valuable insights into the successful facilitation of dynamic load balancing.

4.5: Conclusion

This chapter primarily focused on providing a system description, functional block diagram, and schematic block diagram of a proposed new system architecture for an SD-WAN. In essence, the research aims to investigate the interaction between the end-to-end protocol (TCP) and SD-WAN load balancing algorithms. It is assumed that the introduction of a traffic monitor which carries out thorough packet inspection to provide a new real-time control input to the SD-WAN controller can enhance the ability of the SD-WAN to achieve load balancing across its various physical WAN connections. The next chapter describes how this new SD-WAN architecture has been implemented within a network simulation environment.

CHAPTER FIVE: SYSTEM DESIGN AND IMPLEMENTATION

Chapter five focuses on the practical implementation of the design proposed in chapter four. During the implementation phase of the system design, it is necessary to examine various factors and extensively discuss the specific requirements for all components of the SD-WAN traffic management system, including the controller and monitor.

5.1: Overview

In chapter four a new design of SD-WAN architecture was presented. In that design, key components of the research such as the SD-WAN controller, traffic monitor and various networking devices were presented. Network design holds immense significance for businesses and organisations, serving as a key determinant of performance, resilience, scalability, security, cost, and redundancy of the network (Petryschuk, 2023). Also, the implementation of the proposed SD-WAN architecture considers various factors to ensure adherence to best practices (Dooley, 2023).

5.2: Proposed new SD-WAN Architecture (Functional Block Diagram)

The new SD-WAN architecture is illustrated in Figure 24 and comprises a newly inserted traffic monitor function that acts as an interface between an edge routers control plane and the central SD-WAN controller. End-to-end traffic flows, which in this case are between the client on the left and server on the right, pass over one of the three WAN connections and are actively monitored by the traffic monitor associated with each respective edge router. Using Deep Packet Inspection, these traffic monitors can detect changes in the performance of each traffic flow and should any of them start to experience the onset of congestion, an update can be sent from the traffic monitor to the SD-WAN controller. These updates allow the SD-WAN controller to modify its routing decisions and redistribute traffic flows over the different WAN connections as appropriate. The routing decisions are designed to operate with the mechanism of load balancing which is aimed at even distribution of the load across the network resources. This indicates high effectiveness in achieving several key objectives, including ensuring a high level of quality of service, maximising throughput, preserving network stability, and resource utilisation. The core mechanism behind load balancing is to maintain an equitable distribution of workloads across all connected WAN links continuously. The Deep Packets Inspection enables the capturing of the packet header information of each packet source IP address, destination IP address, source port number, destination port number, protocol type, packet length and window size.

The SD-WAN controller maintains a comprehensive overview of the entire network's resources and aligns them with the specific requirements of applications to enhance network performance. Therefore, applying the concept or principle of load balancing technology in this research is vital to enable multiple-aware routing policies, efficient allocation of network resources, and the attainment of optimal throughput thereby enhancing end-to-end quality of service.

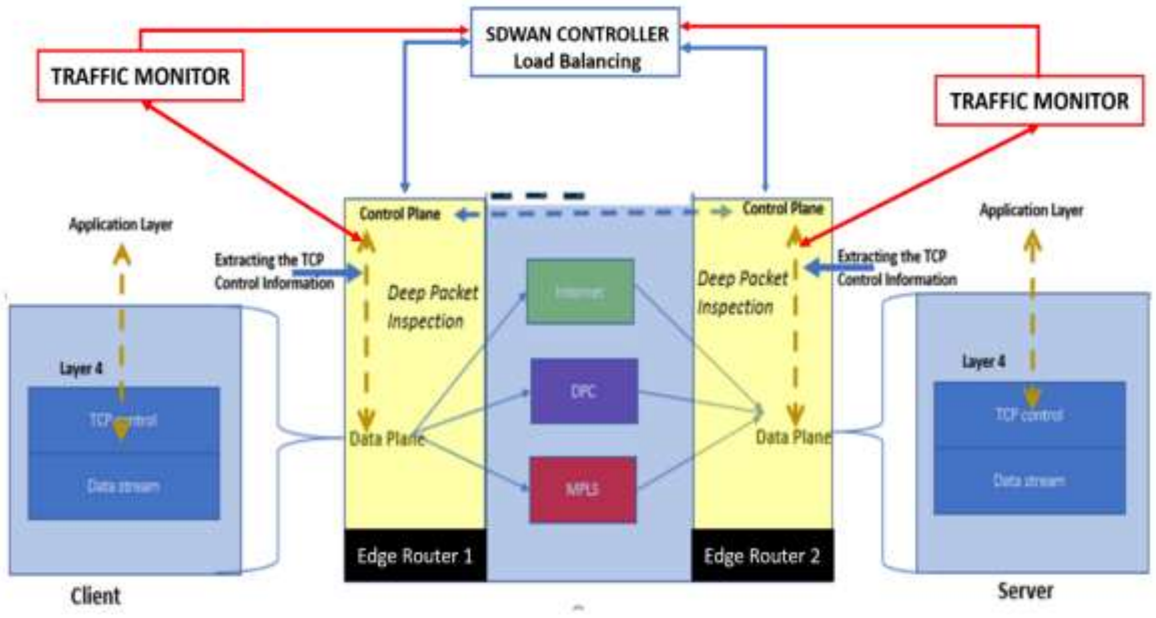


Figure 24: Functional Block Diagram for the research.

5.3: System Description

Figure 25 provides a functional diagram of the operation of the modified SD-WAN architecture. Traffic flows passing the edge router is analysed, classified by the traffic monitor structured in a database for further analysis and filtering. The analysed traffic flows are subsequently passed to the SD-WAN controller, which makes optimal decisions regarding traffic management and based on the routing database. When analysing traffic, if the traffic monitor detects the onset of congestion or, indeed, the recovery from a previously congested state, appropriate traffic flow status updates are sent to the SD-WAN controller. These updates allow the SD-WAN controller to modify its routing policies and distribute modified routing information to the edge routers. As illustrated in Figure 25, there exist two databases. One manages traffic flow to identify congestion, providing an overview and essential details about the traffic flows. The controller's database, on the other hand, holds the routing policies that oversee the entirety of the SD-WAN architecture.

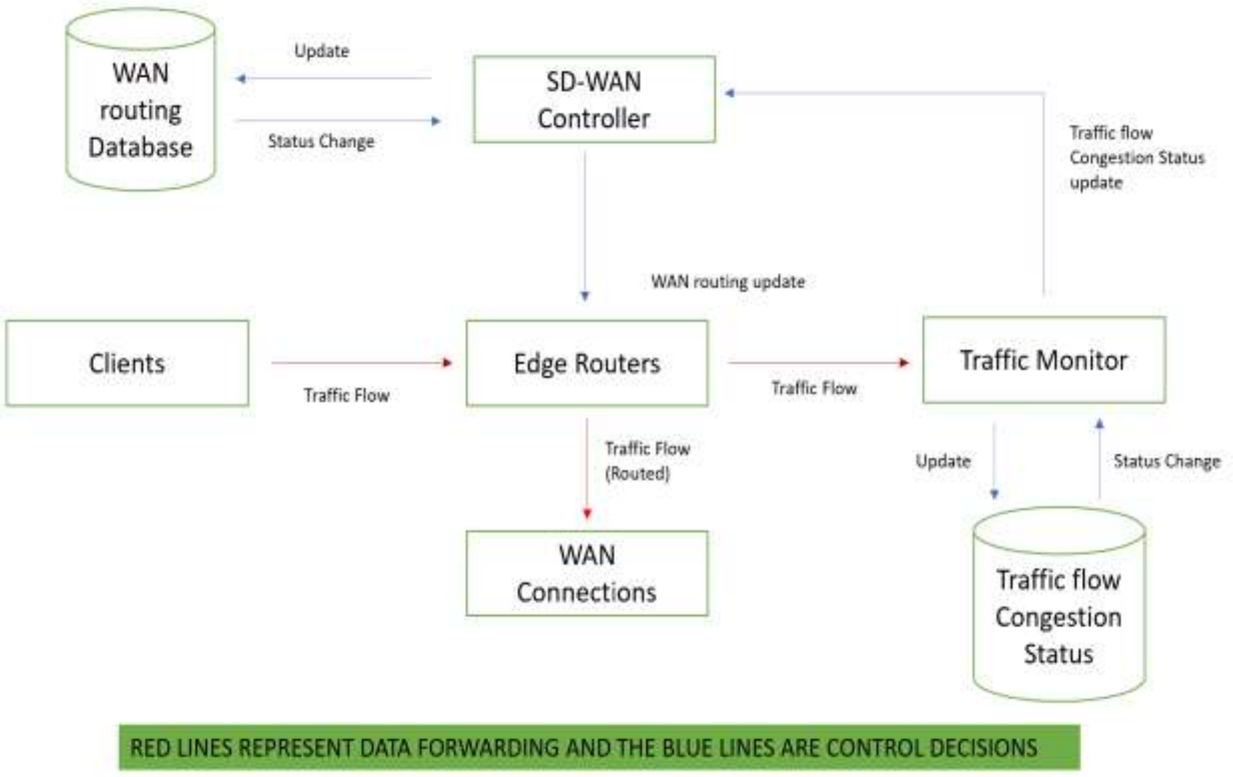


Figure 25: Shows a working operation of the proposed SD-WAN design.

5.4: Schematic Diagram

SD-WAN uses a hierarchical structure known as the three-tier architecture, comprising the data plane, control plane, and management plane. The network design is based on this three-tier architecture methodology, as illustrated in Figure 26, which showcases the different platforms aligned with the three-layer architecture used in the network design. The Data plane within the SD-WAN architecture encompasses the physical network infrastructure required to relay traffic between user applications. In this research, the data plane comprises branch offices which are represented as clients and the headquarters are represented as servers. The control plane is centred around the SD-WAN controller, which acts as the network's intelligence. The SD-WAN controller makes informed decisions regarding the WAN connections over which each traffic flow should be routed. In this layer, the SD-WAN controller is placed alongside the traffic monitor. Finally, the management plane is responsible for defining the overall routing policies governing the use of the various WAN connections.

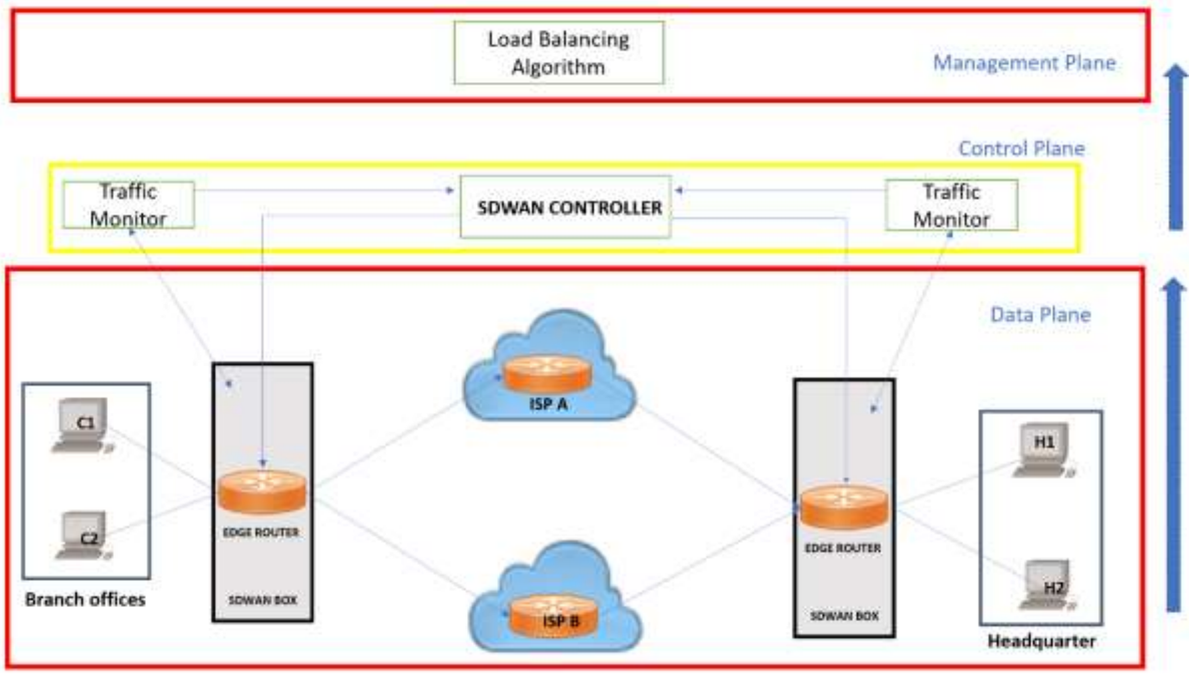


Figure 26: Schematic Block Diagram for the research

5.5: Network Design Using GNS3

GNS3, an open-source network emulator, offers a suitable environment for this research. It supports the utilisation of Docker containers, enabling programmable functionalities like Python programming. Figure 27, which is the network design for traffic management system and SD-WAN controller for the SD-WAN architecture incorporates a hierarchical topology approach. In this hierarchical setup in Figure 27, the clients and server end stations are represented as docker containers labelled as C1 to C6 while the server as S1 and S2 respectively. They are specifically configured to facilitate the transmission of applications or packets. This segment of the network serves as the underlay or access layer. The middle tier focuses on traffic distribution and serves as a boundary regulator between different layers which comprises the edge routers represented as Edge router 1 and Edge router 2. Within this tier, switching, routing, and data filtering are performed among the various devices. The core layer functions as the network's backbone and efficiently routes data between the different WANs of the network design.

The design incorporates core components that consist of devices running the Ubuntu operating system, acting as clients, and connected to a layer 2 switch (SW1). The layer 2 switch is configured to perform various functions, such as VLAN, to ensure the use of a different network ID for each of the clients and servers connected. From the switch (SW1 and SW2), all the client and server devices are connected to the edge router and the traffic monitor via a hub. The Hub's(Hub 2 and Hub 3) between the traffic monitor and edge router are used in the design to allow the traffic monitor to have a comprehensive knowledge of the packet flow passing between the switch and edge routers. The traffic monitor (monitor 1 and monitor 2) operates as an Ubuntu device with programmability capabilities, enabling the capturing, filtering, and analysis of network packets. In this design, two network monitoring devices (traffic monitor) are implemented to capture the traffic flowing in the client/server configuration, which is connected to the SD-WAN controller. The SD-WAN controller, in turn, connects to the distribution tier to implement policies and manage traffic in the network. Also, NAT is used to establish connections between certain devices and the internet, allowing for the updating of programming kernel codes which is easily implemented using the Hub's (Hub 1, Hub 4 and Hub 5).

In reference to Figure 27, the two hubs (Hub 2 and Hub 3) in the L.H.S and R.H.S in the design serves as a device for broadcasting traffic to the traffic monitor. Hence, it is placed in strategic places to implement port mirroring in the GNS3 simulation environment, where the switches do not support this feature(Newberg, 2023; Scarpati, 2023). Furthermore, the switches (SW1 and SW2) are utilised in this design to replicate a physical network, transmitting signals between sources and destinations without participating in the network's routing process. Also, the principle of router on a stick is used to connect the multiple clients to the switch which is in turn connected to the edge routers in the SD-WAN architecture(Shaw, 2023). Also, is the NAT device in the GNS3 environment which acts as a cloud-based connection between appliances and the internet. Also, GNS3 is compatible with a variety of Cisco routers for network design and implementation. The Cisco CSR1000V router is specifically chosen for SD-WAN architecture due to its high performance and compatibility in terms of allowing programmable features into its architecture within the GNS3 emulator. It incorporates network contention control and supports all routing protocols plus it can be easily managed using Cisco or any third-party agent.

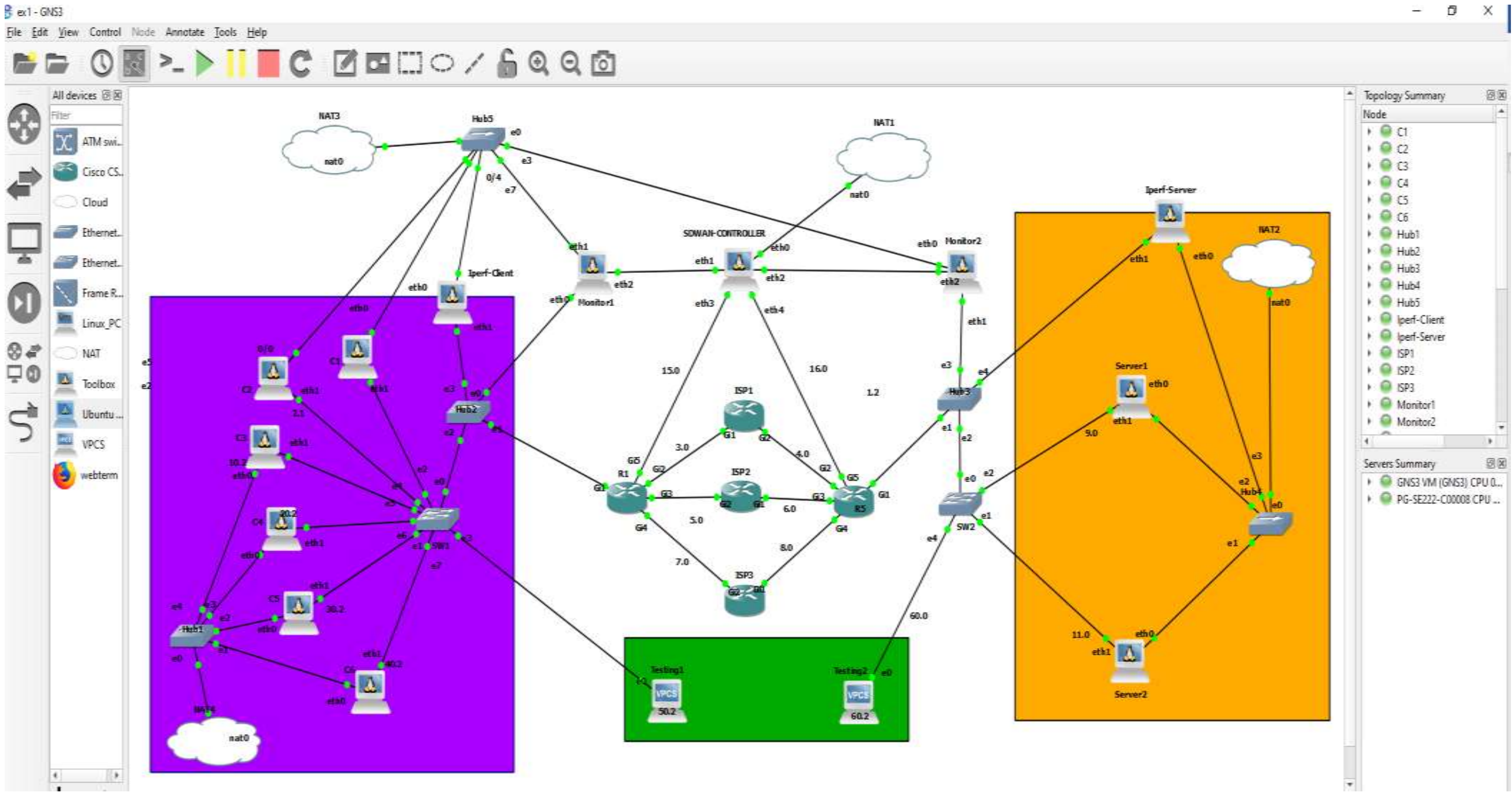


Figure 27: Network Design for SD-WAN Traffic Management

The rationale behind the number of subnets is to ensure ease of IP allocation of the various devices connected to the edge routers. Which will help in simplifying the network and its architecture for better understanding of the network design.

Routers	Interfaces	Network	IP Address	Subnet Mask
Edge Router 1	Gi1	192.168.1.0	Router on a stick	255.255.255.0
	Gi2	192.168.3.0	192.168.3.1	255.255.255.252
	Gi3	192.168.5.0	192.168.5.1	255.255.255.252
	Gi4	192.168.7.0	192.168.7.1	255.255.255.252
	Gi5	192.168.15.0	192.168.15.1	255.255.255.252
Edge Router 2	Gi1	192.168.1.0	Router on a stick	255.255.255.0
	Gi2	192.168.4.0	192.168.4.1	255.255.255.252
	Gi3	192.168.6.0	192.168.6.1	255.255.255.252
	Gi4	192.168.8.0	192.168.8.1	255.255.255.252
	Gi5	192.168.16.0	192.168.16.1	255.255.255.252
ISP 1	Gi1	192.168.3.0	192.168.3.2	255.255.255.252
	Gi2	192.168.4.0	192.168.4.2	255.255.255.252
ISP 2	Gi1	192.168.6.0	192.168.6.2	255.255.255.252
	Gi2	192.168.5.0	192.168.5.2	255.255.255.252
ISP 3	Gi1	192.168.8.0	192.168.8.2	255.255.255.252
	Gi2	192.168.7.0	192.168.7.2	255.255.255.252

Table 7: Network and IP address Configuration of the Network Design

5.6: Client’s (Branch Offices) and Server’s (Headquarters)

The implementation has been designed to include six clients in the branch office, each of which is connected to three different WANs or ISPs. To create a realistic network environment, each client has been configured with individual network addresses. Both the branch offices and the headquarters employ the 'router on a stick' concept to establish connections to the edge routers, as described in Figure 27. Table 7 provides an overview of the network address and the corresponding IP addresses for all branch offices.

Branch Office	IP addresses	Gateway IP Address	Network	VLANS
Client 1	192.168.1.2	192.168.1.1	192.168.1.0	200
Client 2	192.168.2.1	192.168.2.2	192.168.2.0	100
Client 3	192.168.10.2	192.168.10.1	192.168.10.0	20
Client 4	192.168.20.2	192.168.20.1	192.168.20.0	40
Client 5	192.168.30.2	192.168.30.1	192.168.30.0	10
Client 6	192.168.40.2	192.168.40.1	192.168.40.0	30
Headquarter	IP addresses	Gateway IP Address	Network	VLANS
Server 1	192.168.9.2	192.168.9.1	192.168.9.0	300
Server 2	192.168.11.2	192.168.11.1	192.168.11.0	400

Table 8: Summary of Client's and Server's network configuration

Within the branch office, client systems execute Python scripts to generate TCP traffic. The program responsible for generating this traffic is available in the GitHub repository, along with other Python code segments developed as part of this research. To generate this traffic, the 'Scapy.all' library has been used. This python library plays a crucial role in creating a variety of packets that can be sent and captured through wired connections. The number of packets generated is randomised to enhance the code's flexibility. Particular attention has been given to the parameter of window sizes and packet lengths as represented by the pseudocode shown in Figure 28.

```
Start
|
|-- Import Libraries
|
|-- Initialize Destination IP Addresses
|
|-- Create a Socket (client_socket)
|
|-- Set Server Address (server_address)
|
|-- Try Block Start
|   |
|   |-- Connect to Server
|   |   |
|   |   |-- If Successful
|   |   |   |
|   |   |   |-- Print "Connected to server_address"
|   |   |
|   |   |-- Send Packets in a Loop (1000 times)
|   |   |   |
|   |   |   |-- Randomly select a destination IP address
|   |   |   |
|   |   |   |-- Generate random values for packet length and window size
|   |   |   |
|   |   |   |-- Create a TCP packet with custom source IP, random destination IP, packet length, and
window size
|   |   |   |
|   |   |   |-- Send the packet
|   |   |   |
|   |   |   |-- Pause for 0.1 seconds
|   |   |
|   |   |-- Print "Packets sent successfully!"
|   |
|   |-- Finally Block
|       |
|       |-- Close Socket (client_socket)
|
|-- End
```

Figure 28: Simple pseudocode for TCP Packet Generation for Client

The same principle is used in implementing the headquarter office with two server devices connected in different networks. The server code is designed to receive the packets from the clients using the socket connection approach which is developed using python scripts. Figure 29 is a pseudocode used within these servers.

```
Start
|
|-- Initialize server_socket as a TCP/IP socket
|
|-- Bind server_socket to IP address '192.168.9.2' and port 8000
|
|-- Start listening for incoming connections on server_socket
|
|-- Set packet_count to 0
|
|-- Infinite Loop
|   |
|   |-- Print "Waiting for a connection..."
|   |
|   |-- Accept an incoming connection, resulting in client_socket and client_address
|   |
|   |-- Try
|   |   |
|   |   |-- Infinite Loop
|   |   |   |
|   |   |   |-- Receive data on client_socket with a buffer size of 1024 bytes into data
|   |   |   |
|   |   |   |-- If data is empty (indicating the client has closed the connection)
|   |   |   |   |
|   |   |   |   |-- Exit the loop
|   |   |   |
|   |   |   |-- Increment packet_count
|   |   |   |
|   |   |   |-- Print "Received packet {packet_count}: {data decoded as text}"
|   |   |
|   |-- Finally
|   |   |
|   |   |-- Close the client_socket
|
|-- End
```

Figure 29: Simple pseudocode for TCP Packet reception for Server

5.7: Traffic Monitor

The traffic monitor is one of the key components of this SD-WAN architecture and is responsible for capturing and analysing the traffic within the architecture. It operates as an application that autonomously collects and summarises traffic flows, from which updates are sent to the SD-WAN controller for routing updates. In this design, a combination of the port-based approach and Deep Packet Inspection is employed. The port-based approach focuses on classifying packets based on their TCP and UDP port numbers, but it has limitations such as the inability to classify protocols like Peer-to-Peer (P2P) or passive FTP due to the use of dynamic ports. By incorporating Deep Packet Inspection alongside the port-based approach, these limitations can be addressed, allowing for a more comprehensive and accurate analysis of packet flows.

5.7.1: Requirements

The Traffic Monitoring system is developed to capture and analyse network activity using programming tools such as Python programming language. In this implementation, an Ubuntu OS-based docker container is installed within the GNS3 VM and seamlessly integrated into the GNS3 GUI. Aligned with the proposed SD-WAN architecture, the docker container is connected to the NAT cloud to facilitate the download of applications (applications such as Pyshark, SQLite, PiP, time and many more) that support the complete functionality of capturing and monitoring traffic. Key parameters utilised in this system include the Pyshark library for capturing network packets and storing relevant data in an SQLite database. The traffic monitoring system is designed with a focus on three essential characteristics: efficiency, non-interference, and comprehensiveness. Efficiency pertains to optimising resource utilisation in traffic monitoring, considering the significant volume of network traffic generated. Non-interference refers to the monitoring process being designed not to disrupt management processes like packet forwarding from end hosts, particularly in network environments with limited resources. The monitoring system is envisioned as a self-driven network, necessitating a comprehensive understanding of traffic flow, encompassing aspects such as capturing and classifying various parameters contained within packets.

5.7.2: Design

The Traffic Monitor is designed to be a lightweight monitoring device that is programmable based on defined policies aimed in capturing and forwarding traffic in the network. A simple architecture of a self-driven network is shown in Figure 30.

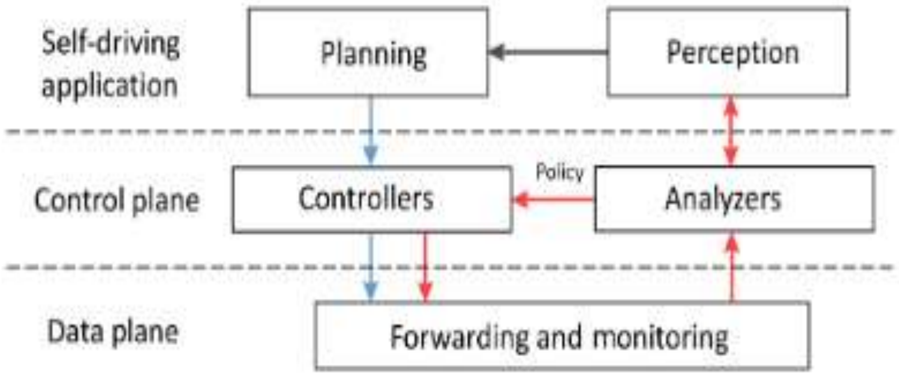


Figure 30: Simple architecture of a Self-Driven Network (Hu et al., 2021)

Figure 30: illustrates a self-driven network where the data plane incorporates both the forwarding and monitoring components. In the control plane, the emphasis lies on the analysers that provide policy feedback to the controllers, while the application layer encompasses perception and planning. The Traffic Monitor function implemented in this research operates by leveraging the native kernel of Linux using the package "pyshark" to capture and analyse packets. Figure 31: presents the architecture of the Traffic Monitoring system within the context of the SD-WAN architecture. The architecture enables the capture of data through interfaces connected to the network. Subsequently, Linux Socket Filtering (LSF) and Packet classification mechanisms come into play. The packets are then analysed and classified by the analyser and classifier, which generate updates sent to the controller. These routing updates are represented as Flow ID's which are forwarded to the controller and have key components such as the source address, destination address, source/destination port and congestion flags.

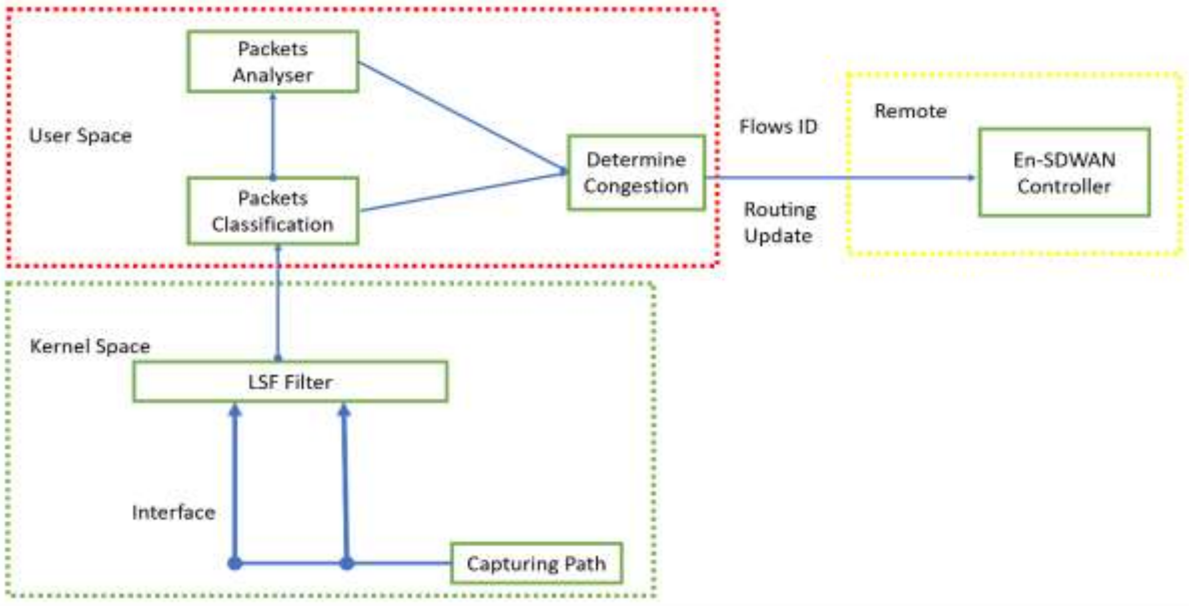


Figure 31: Shows the proposed architecture of the Traffic Monitor for the SD-WAN architecture.

Table 9: shows the summary of the IP configuration of the monitor function in the SD-WAN architecture.

Monitoring Function IP Allocation			
Monitor Function 1			
Interface	IP addresses	Network	Subnet Mask
Eth0	192.168.0.2	192.168.0.0	255.255.255.0
Eth1	192.168.242.175	192.168.242.0	255.255.255.0
Eth2	192.168.100.1	192.168.100.0	255.255.255.0
Monitor Function 2			
Eth0	192.168.242.172	192.168.242.0	255.255.255.0
Eth1	192.168.11.2	192.168.1.0	255.255.255.0
Eth2	192.168.200.1	255.255.255.0	255.255.255.0

Table 9: Summary of the IP configuration/Allocation of the monitor function in the SD-WAN architecture.

5.7.3: Capturing

Capturing represents the initial phase of implementing the traffic monitor, which involves capturing packets using the native kernel space. The traffic monitor system's design follows the standards utilised by TCP-Dump and Wireshark. Among the suitable interfaces for packet capture, the monitoring system employs PyShark, a Python wrapper for the TShark command-line utility, which is part of the Wireshark network analysis suite. PyShark offers a more convenient way of working with network packets compared to directly using the underlying libpcap library. By leveraging Wireshark/TShark dissectors, PyShark simplifies the process of packet capture by providing an object-oriented interface for dissecting and analysing packets. It's important to note that while Libpcap serves as a lower-level library for packet capture, PyShark acts as a higher-level. Although other options in Libpcap, such as snapshot length, promiscuous mode, monitor mode, and packet buffer timeout, are available, the immediate mode is selected for its dynamic characteristics (McCanne, 2023). The immediate mode is used because it creates a platform for real time packet analysis and the libcap captures directly without storing them to a file. The Traffic Monitoring system functions dynamically by applying a lightweight filtering process, utilising Linux Socket Filtering (LSF), to all captured packets. LSF is chosen due to its performance and operational resemblance to Berkeley Packet Filtering (BPF) (Jay Schulist, 2023). The BPF is the method used in Wireshark and is like the

LSF which is chosen for the implementation of this research. The implementation of both PyShark and Linux Socket Filtering is achieved using Python programming scripts in the GNS3 emulation environment, which is achieved by using docker.

5.7.4: Packets Classifications

Packet classification is an integral area of this research which is implemented after the capturing of the packets using the *Eth0* interface. Figure 32: shows a simple description of the packet's classification of packets.

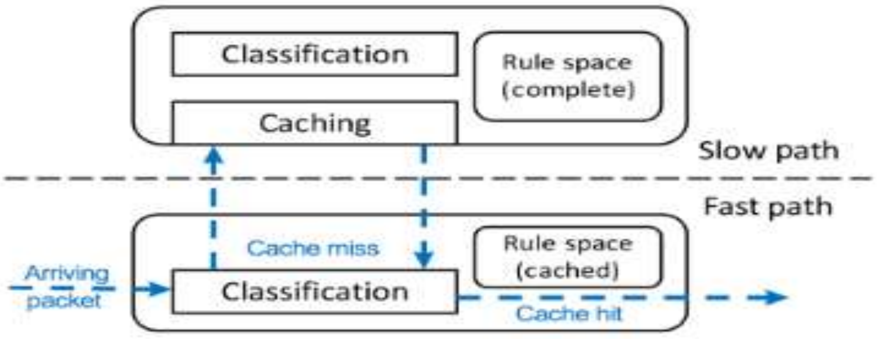


Figure 32: Simple Classification architecture of Monitored Packets (Hu et al., 2021).

In reference to Figure 32: the captured packets in the traffic monitor system operate in both fast and slow path for the classification of the packets. Packets that are not classified in the fast path are classified in the slow path based on the classification mechanism. Traffic Monitoring system divides its working operation into monitoring and forwarding policies, the monitoring policies are achieved in the capturing stage while the forwarding is achieved by sending congested updates to the SD-WAN controller remotely using the socket principle.

Table 10 lists the parameters selected for the network analysis and are used in forming the database for the classification of the packets captured and analysed using python scripts in the traffic monitor. The changes in the TCP window, packet length, total bytes, and protocol offer valuable indications of congestion levels within the network. The TCP window size denotes the maximum quantity of packets permissible for transmission, with a reduced window size indicating potential network congestion. Monitoring total bytes transmitted offers insights into overall network utilisation, indicating heavier traffic loads and potential congestion. Also, different protocols behave differently in response to congestion, with TCP adjusting transmission rates based on congestion level while UDP does not. Examining these parameters enables network administrators to assess congestion levels, identify bottlenecks, and implement strategies for optimising network performance and alleviating congestion issues.

The congestion level is used to indicate the congestion flag based on the traffic flows between the clients and servers which is an integral part of the data captured or examined. Timestamp is important to examine the average latency and percentage of packet loss which can be used in determining the level of congestion in the network. This information is applied to make decisions about the network performance as presented in the database.

Network Parameters	Description
ID Integer	Database number entry
Timestamp	Indicate the time for capturing the packets
Source_IP_Address	Source Internet Protocol address
Source_Port_Number	Source Port number
Dest_IP_Address	Destination Internet Protocol address
Dest_Port_Number	Destination Port number
Length	Length of the Packet
Congestion_level	Indicate the level of Congestion
IP_Protocol	Packets Protocol
Total_Bytes	Total bytes of Packets Transmitted

Table 10: Database summary of network parameters captured and analysed.

The classification of the incoming packets is likened to the concept of supervised learning machine learning algorithms where the packets can be classified to a specific category or class which focus on a combination of the key parameters. In the implementation of this methodology, there is a need for a filter database to send routing updates to the SD-WAN controller for decision making. Table 11: show a summary of the database captured and analysed, it includes a summary of the vital items mentioned in Table 10.

Src_Add	Src_Port	Src_Wn	Dst_Add	Dst_Port	Dst_Wn	Length	Congestion Level	Action
192.168.1.10	44394	501	192.168.2.10	5001	502	1783.0	High	Reroute
192.168.5.10	44394	603	192.168.6.10	5001	604	1832.4	Low	Steady
192.168.10.10	N/A	55127	192.168.11.10	N/A	55128	6084.7	Medium	Reroute
192.168.20.10	N/A	44713	192.168.21.10	N/A	44714	7042.6	Low	Steady

Table 11: summary of the database captured in the SD-WAN traffic management tool.

In reference to Table 11, action is taken to a flow or entry based on the level of congestion, but the action is taken at the controller level and not at the monitoring stage. The level of congestion is determined by using the relationship of window size, percentage of packet loss and average latency. The thresholds of congestion are defined as “High”, “Medium”, “Low” and “no congestion”.

Thus, whenever there is a change in the status of the congestion level or action, a summary or an update is sent to the controller using a function developed in the python program. The summarised update (flow_id) sent to the controller comprises key parameters such as the "Source IP, Destination IP, Source Port, Destination Port, Congestion level, and action". Also, it is important to pass this summary information to the controller but only when there are changes with the congestion level and action is required to be taken. The function operates by sending the packet information in JSON (JavaScript Object Notation) format to the SD-WAN controller (decision-making container) using an HTTP Post request using the specific URL (Uniform Resource Locator) that is assigned to the controller. A simple illustration of the information sent to the controller is shown in the Table 12 below, this information sent to the controller is a summarised version of the traffic monitor database. It is important to note that the controller information is summarised into only the details of changes with the congestion level because of achieving maximum performance of the SD-WAN controller.

Src_Add	Src_Port	Dst_Add	Dst_Port	Congestion Level
192.168.1.10	44394	192.168.2.10	5001	High
192.168.5.10	44394	192.168.6.10	5001	No Congestion
192.168.10.10	N/A	192.168.11.10	N/A	Medium
192.168.20.10	N/A	192.168.21.10	N/A	Low

Table 12: Summarised version of controller information

5.7.5: Python Programming

Python programming tool is used for the programming features of the network monitoring system. This is implemented in the docker container using an Ubuntu - Linux operating system in the GNS3 emulation environment. In reference to Figure 27, the two-traffic monitor is connected to three interfaces which includes a NAT connection, edge router connection and the SD-WAN controller. The NAT connection focuses on the provision of internet connectivity to the device for the installation of the various Python modules (Pyshark, SQLite, PiP, time and many more). During the implementation of the Python scripts, essential libraries such as Pyshark, SQLite, time, and statistics are utilised. These libraries enable functions like packet capture, database management, time delay, and mathematical computations. The program functions as a network monitoring tool by capturing and filtering a continuous stream of network packets, which are then stored in an SQLite database. The captured packet information is analysed by extracting key details, determining congestion levels, and ensuring that similar flow characteristics are not duplicated in the database. Also, the program evaluates congestion by examining window size, percentage of the loss packets and average latency aiming to send the analysed packets to an SD-WAN controller. A simple pseudocode in the form of a flowchart describing all the python code is shown in the textbox in Figure 33.

```
Start
|
|- Create Database and Table
|
|- Initialize Variables and Headers
|
|- Start Packet Capture
|
|--- Loop:
|    |
|    |--- Capture the Network Packet
|    |
|    |--- Check if TCP Packet is available.
|    |   |
|    |   |--- Extract Packet Information
|    |   |
|    |   |--- Calculate Congestion Level
|    |   |
|    |   |--- Check if Packet Exists in Database
|    |   |   |
|    |   |   |--- If Packet Exists
|    |   |   |   |
|    |   |   |   |--- Update Packet Record in Database
|    |   |   |   |--- Print Updated Record
|    |   |   |   |
|    |   |   |--- If Packet Does Not Exist
|    |   |   |   |
|    |   |   |   |--- Insert Packet Record into Database
|    |   |   |   |--- Get New Record ID
|    |   |   |   |--- Fetch New Record from Database
|    |   |   |   |--- Print New Record
|    |   |   |
|    |   |--- Commit Changes to Database
|    |
|    |--- If Non-TCP Packet
|    |   |
|    |   |--- Print "Non-TCP Packets"
|
|- Analyse Congestion
|    |
|    |--- Calculate Packet Loss Percentage
|    |
|    |--- Calculate Average Latency
|    |
|    |--- Determine Congestion Level (Packet loss Percentage, Average Latency and Window Size)
|    |
|    |--- Send Analysed Information to En-SDWAN Controller
|    |
|    |--- Print Congestion Analysis Results
|
|- Close Database Connection
|
End
```

Figure 33: Pseudocode of the Traffic Monitoring system

5.7.6: Mathematical function for Traffic Monitor:

Mathematical function or modelling is used in this research to describe the python programming codes into mathematical concepts which can be used for further analysis and verification of the concept. The mathematical model clearly describes in simple steps the various functionalities and dictionaries used in the implementation of the traffic monitor and En-SDWAN controller. The mathematical model for the monitor is shown below in Figure 34.

Variables:

- database: A database to store packet information.
- table: A table in the database to store packet information.
- packet_buffer: A buffer to store captured packets.
- packet_information: A dictionary to store the information of a packet, such as the source IP address, destination IP address, packet length, and congestion level.
- packet_loss_percentage: The percentage of packets lost.
- average_latency: The average latency of packets.
- congestion_level: The congestion level of the network.

Functions:

- create_database_and_table(): This function creates the database and table to store packet information.
- initialize_variables_and_headers(): This function initializes the variables and headers used in the network traffic congestion detection process.
- start_packet_capture(): This function starts capturing packets from the network.
- capture_the_network_packet(): This function captures a packet from the network and stores it in the packet buffer.
- check_if_tcp_packet_is_available(): This function checks if the packet in the packet buffer is a TCP packet.
- extract_packet_information(): This function extracts the information of a packet from the packet buffer and stores it in the packet_information dictionary.
- check_if_packet_exists_in_database(): This function checks if the packet information in the packet_information dictionary exists in the database.
- update_packet_record_in_database(): This function updates the packet record in the database with the new packet information.
- insert_packet_record_into_database(): This function inserts the new packet information into the database.
- get_new_record_id(): This function gets the ID of the new packet record inserted into the database.
- fetch_new_record_from_database(): This function fetches the new packet record from the database.
- commit_changes_to_database(): This function commits the changes to the database.
- analyse_congestion(): This function analyses the congestion of the network based on the packet_loss_percentage, window size, and average_latency.
- calculate_packet_loss_percentage(): This function calculates the packet loss percentage based on the number of packets lost and the total number of packets.
- calculate_average_latency(): This function calculates the average latency based on the latencies of all packets.
- determine_congestion_level(): This function determines the congestion level of the network based on the packet_loss_percentage, window size, and average_latency.
- send_analysed_information_to_decision_making_container(): This function sends the analysed information about the network congestion to the decision-making container.
- print_congestion_analysis_results(): This function prints the congestion analysis results to the console.
- close_database_connection(): This function closes the database connection.

Figure 34: Mathematical Function for Monitor Program

5.8: En-SDWAN Controller

For this research, the proposed SD-WAN controller will be represented as En-SDWAN controller where the prefix “En” is referred to as Enhanced. The En-SDWAN controller is the heart of the SD-WAN architecture presented in this research design, because it makes informed or intelligent decisions about the distribution of traffic flows over the various WANs in the SD-WAN architecture. It is designed to operate as a self-driven application that can interact with the edge routers for load sharing of traffic flows. In reference to Figure 35: the En-SDWAN controller takes input values (congestion updates) from the two traffic monitors. Also, it is configured with three interfaces that connect to the NAT for internet connectivity and to the edge routers were forwarding of route updates to the network is established. Furthermore, the En-SDWAN controller is built based on a lightweight container in the GNS3 emulation environment and various packages are installed to ensure the smooth functionality of the device. Figure 38 provides a simplified representation of the working model of the En-SDWAN controller. In this illustration, the input values or information are obtained from two traffic monitors. These values are then transmitted to the En-SDWAN controller, which subsequently communicates with the Edge routing devices. The edge routing devices implement routing updates to the underlay networks as shown in the Figure 35.

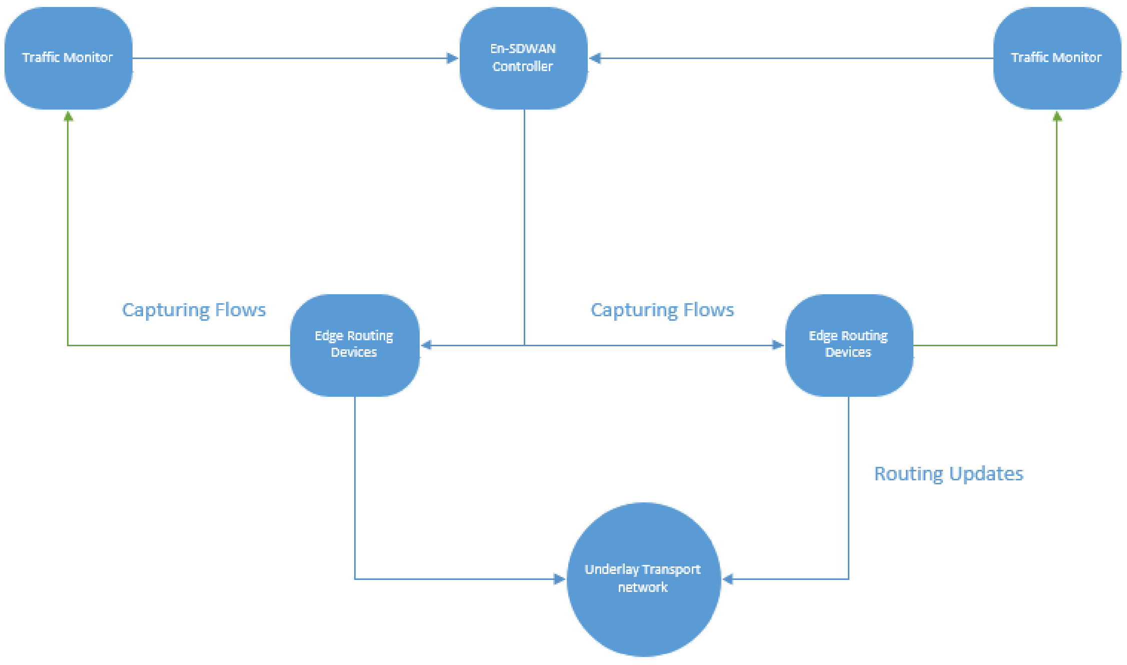


Figure 35: Illustration of SD-WAN Controller with the network architecture.

5.8.1: Requirement

The En-SDWAN controller is implemented to have a comprehensive knowledge of the summarised network traffic flows to actively act when congestion is detected. Performance of the entire network is integral based on the policies of the En-SDWAN controller, this will ensure dynamic adaptation of changes in network traffic and congestion. The En-SDWAN controller communicates directly with the edge routers that need to be managed and the nature of the applications being handled by the controller. Based on the SD-WAN architecture, two edge routers are managed by the En-SDWAN controller. In terms of application and working model of the routing flow, the proactive, reactive and hybrid are three fundamental application types used for software defined networking controller development. The reactive approach constantly led to changes and adjustment of the routing flow tables because of dynamic flows or changes in the network behaviour (Salisbury, 2023). Figure 36: illustrates the reactive approach for software defined networking applications.

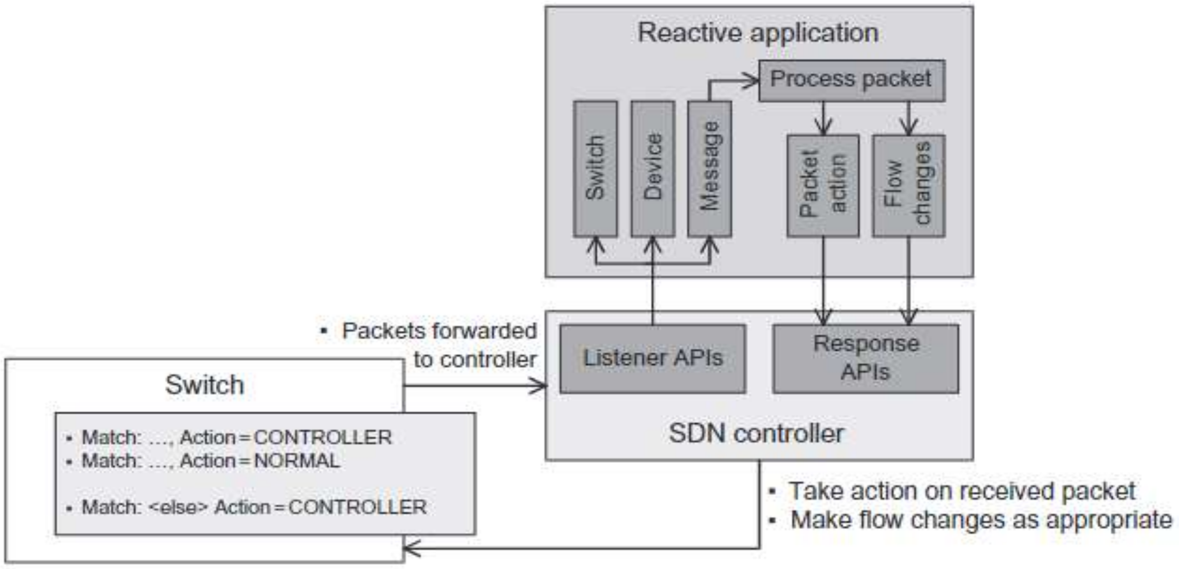

Reactive application
Process packet
Switch
Device
Message
Packet action
Flow changes
• Packets forwarded to controller
Listener APIs
Response APIs
SDN controller
Switch
• Match: ..., Action=CONTROLLER
• Match: ..., Action=NORMAL
• Match: <else> Action=CONTROLLER
• Take action on received packet
• Make flow changes as appropriate

Figure 36: The reactive approach for software defined networking applications (Paul Goransson, Chuck Black, 2016).

In figure 36: when a traffic flow is sent from the device, it will be sent to the packet listener and when no match for the flow is found. The controller creates an OpenFlow packet-in packet and sends it to the controller for action. The operation of the proactive is different from the reactive. The proactive approach to the controller flow table has been populated before the arrival of packets. Hence, the traffic captured from the network via the hub sent to the controller is sent at line rate and won't require packet-in situation (Salisbury, 2023). Figure 37: shows a proactive approach for software defined networking controller approach.

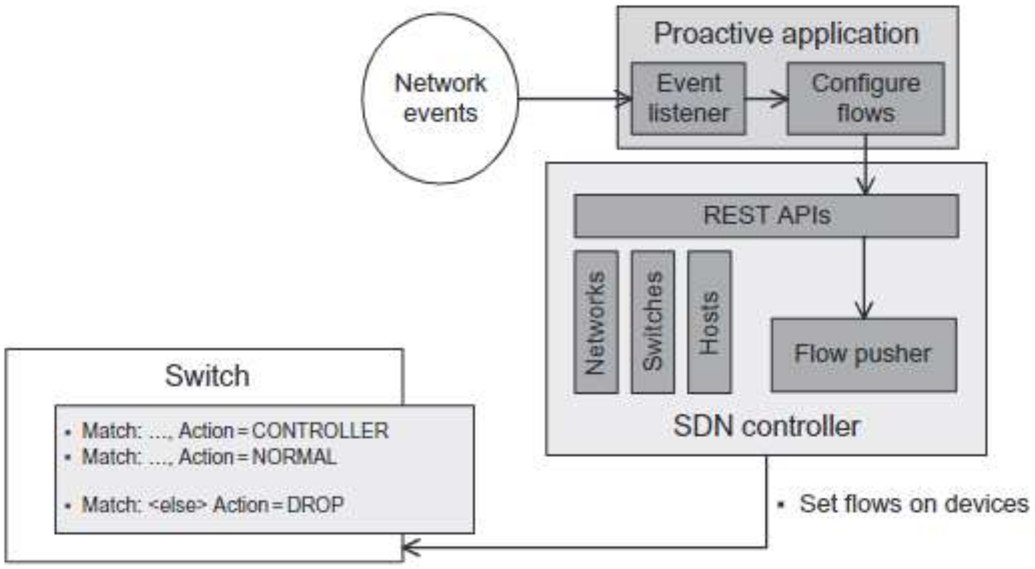


Figure 37: The proactive approach for Software Defined Networking Applications (Paul Goransson, Chuck Black, 2016).

The hybrid approach is a combination of the proactive and reactive method. It creates flexibility and sets a granular traffic control and provides low latency forwarding for the entire traffic.

In this research, the hybrid method is applied in the implementation of the En-SDWAN controller by ensuring analysed and summarised network traffic flows are given actions based on the network conditions. Hybrid model of En-SDWAN controller is applied because the monitoring stage performs a proactive approach where packets are monitored, analysed, and customised into a database. A summary of these flows that are congested arrived at the controller and the controller used a reactive model to react or trigger actions to be taken based on the flows that have arrived at the controller. Hybrid approach has been implemented for the En-SDWAN Controller because of the flexibility and dynamic nature of the proposed controller.

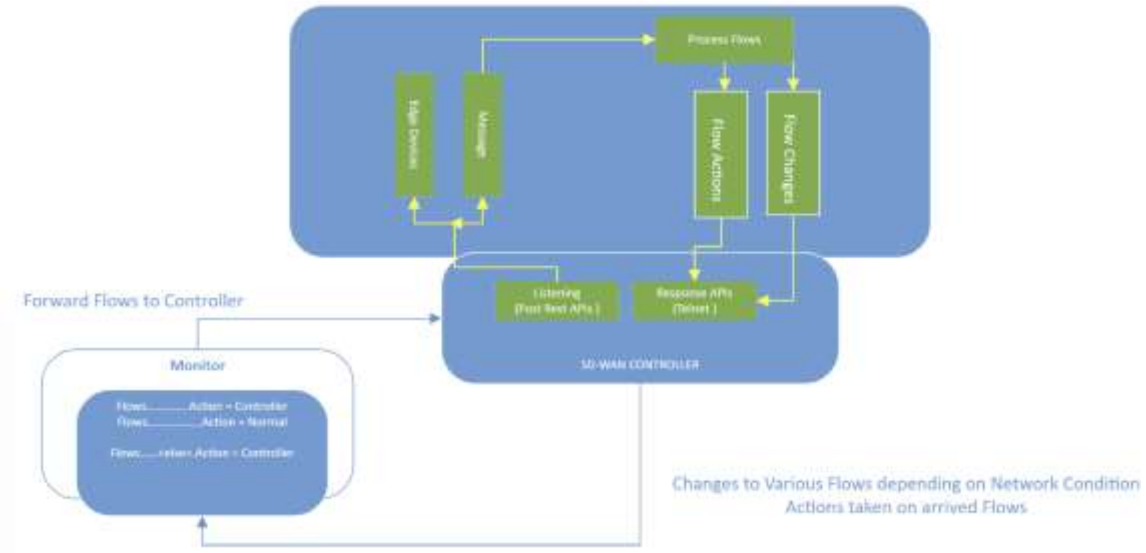

Process Flows
Edge Devices
Manage
Flow Actions
Flow Changes
Listening (Post Rest APIs)
Response APIs (Telnet)
SD-WAN CONTROLLER
Forward Flows to Controller
Monitor
Flows.............Action = Controller
Flows.............Action = Normal
Flows......<else> Action = Controller
Changes to Various Flows depending on Network Condition
Actions taken on arrived Flows

Figure 38: The Hybrid Approach for Software Defined Networking Applications (Paul Goransson, Chuck Black, 2016).

5.8.2: Design

The En-SDWAN controller is designed to intelligently control the network traffic via the edge routers. A simple architecture of the SD-WAN controller is shown in Figure 39.

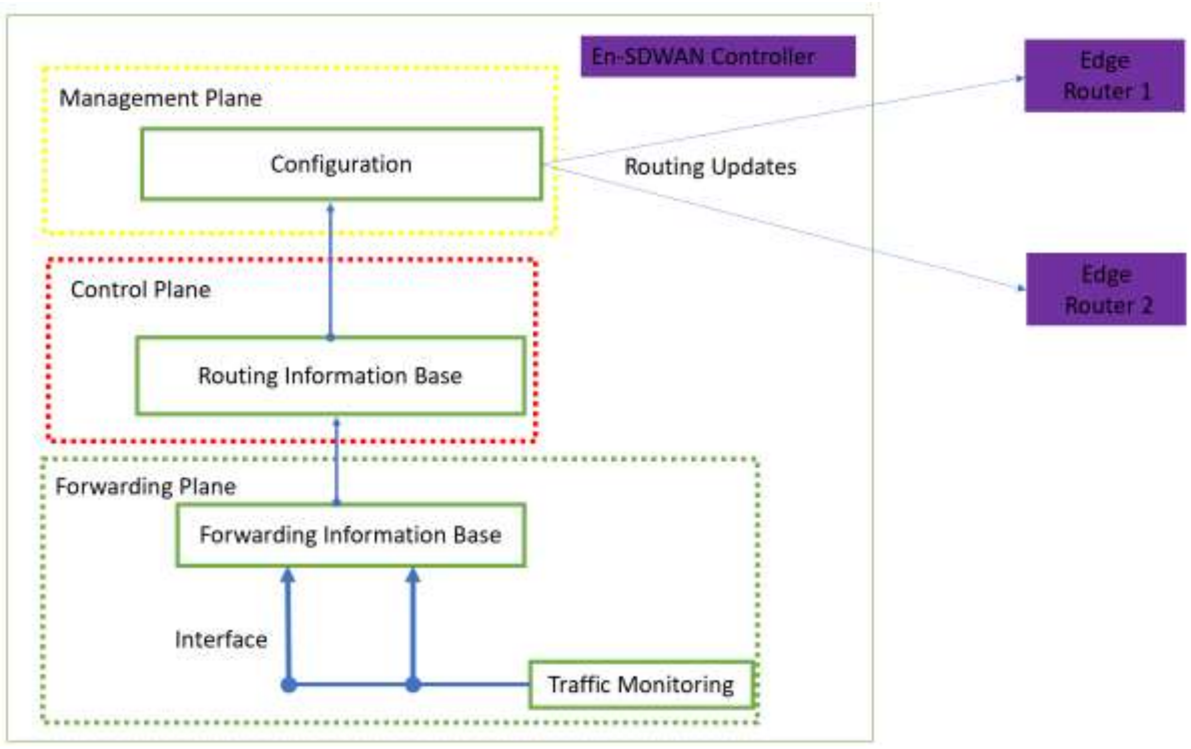


Figure 39: Architecture of the En- SD-WAN controller.

The En-SDWAN controller acquires summarised traffic analysis from the traffic monitor, which is then represented as a Forwarding Information Base (FIB) in the control plane. The FIB contains the summary information obtained from the traffic monitor. The Routing Information Base (RIB) plays a role in managing and programming the forwarding information base for various flows or events. It acts as an intermediary, facilitating communication between the configuration settings and the forwarding information base. Within the RIB, there is a mini database that stores the existing routing information, indicating the various flows from the client to the server. To achieve routing and changes to route flows, programming measures are utilised. The final stage involves configuring settings that focus on routing policies, such as static routes. These routing updates are then shared with the edge routers for updating and automating the network's routes. Based on the network architecture represented in Figure 27, the En-SD-WAN controller is connected to various interfaces as shown in Table 13.

En-SD-WAN controller			
Interface	**IP addresses**	**Network**	**Subnet Mask**
Eth0 (NAT)	192.168.242.175	192.168.242.0	255.255.255.0
Eth1	192.168.100.2	192.168.100.0	255.255.255.0
Eth2	192.168.11.1	192.168.11.0	255.255.255.0
Eth3	192.168.15.2	255.255.255.0	255.255.255.0
Eth4	192.168.16.2	255.255.255.0	255.255.255.0

Table 13: Enhanced SDWAN controller IP Address Allocation

5.8.3: En-SDWAN Controller Method

The initial step entails configuring static IP routing commands in the edge routers. This configuration facilitates default routing between the client and the server, determined by the traffic policies. The approach is derived from the principle of mapping different WANs to traffic flows, aiming for an equitable distribution of these flows. Table 14 provides an outline of the configuration for diverse WANs, contingent on traffic patterns, link capacity, and allocated bandwidth.

WAN LINK	WAN Network	Traffic Flows	Link Capacity (Mbps)	Bandwidth (Mbps)
ISP 1 – WAN 1	192.168.3.0: 192.168.4.0	TF6	100	55
ISP 2 – WAN 2	192.168.5.0: 192.168.6.0	TF1	100	30
		TF3		35
ISP 3 – WAN 3	192.168.5.0: 192.168.6.0	TF2	100	30
		TF4		23
		TF5		25

Table 14: En-SDWAN Controller Algorithm configuration

Referring to Table 14, it's vital to emphasise that initial assumptions are made about link capacities, and these can be fine-tuned to align with the specific scenario. The data illustrated in Table 14 underscores critical factors essential for effective traffic flow management principles, particularly link capacity, utilised capacity, and unused capacity for each WAN link. To ensure an effective traffic balance, the link capacity is set at 100. This implies that, for optimal traffic distribution in this design, traffic should not exceed 80% of the link's capacity, which is the target implemented.

In cases where WAN link traffic volume surpasses 80% of its bandwidth, alternative WAN links should be considered. The selection of an alternative link is determined by the available bandwidth, calculated by the difference between the total link capacity and the used link capacity based on bandwidth provisions. If the current link doesn't meet this bandwidth condition, the next available link undergoes the same evaluation to ensure even traffic distribution. In situations where all conditions are tested and WAN links operate at full capacity, packet flows need to remain on the congested link until congestion gradually diminishes. This entire concept is implemented through Python programming, as described in the upcoming sections.

Furthermore, the traffic flow mapping across various WAN connections influenced the decision to implement static IP routing. Six client applications from the branch office, represented as flow1 to flow6, are mapped to the three WANs based on bandwidth size and a desire for balanced traffic distribution among the WANs. Initially, traffic flows consistently and smoothly amongst the WANs, minimising congestion. However, in cases of congestion, traffic flows can be redirected from one WAN connection to another to maintain network stability and flow. The movement of traffic flows demonstrates the facilitating of the dynamic load balancing in the SD-WAN by reacting to changes in any congestion experienced by each traffic flow.

5.8.4: Python Programming

The implementation of the En-SDWAN controller method is achieved using various python libraries ranging from socket, paramiko, time and statistics. Each of these libraries or modules are used for specific functions in the performance of the En-SDWAN program with the aim of even distribution of traffic flows and reduction of congestion. A step-by-step breakdown of the code is provided below.

The code begins by importing necessary Python modules:

- *`socket`: Allows socket-based communication, crucial for network operations.*
- *`time`: Provides functions for adding delays, essential for controlling the execution flow.*
- *`paramiko`: A library for SSH communication and working with network devices.*
- *`telnetlib`: A module for Telnet communication with network devices.*
- *`defaultdict`: Provides a dictionary-like data structure with default values.*

This is followed by a definition of a Python class named `SDWANController` encapsulating the functionalities essential for SDWAN traffic management. The class constructor (`__init__`) initialises the controller with specified parameters:

- *`monitor_ip`, `monitor_port`: IP address and port for monitoring.*
- *`router_ips`: List of router IP addresses.*
- *`router_username`, `router_password`, `router_enable_password`: Authentication credentials.*

Then, a method is defined which initialises the routing table, setting default routes and capacities for various traffic flows. This is followed by Implementing methods to update routing tables on routers using either Telnet or Paramiko, allowing for dynamic routing changes and defined a method to handle congestion events and reroute traffic flows, accordingly, optimising traffic distribution across WAN links based on defined logics. A simple flow chart summarising the flow of the program is shown in Figure 40.

```
Start
|
|--- Initiate En-SDWANController
|   |
|   |--- Initiate routing tables.
|   |   |
|   |   |--- Default WANs capacity
|   |   |--- Setup initial routing on routers
|   |   |
|   |   |--- Loop (Process Congestion Events)
|       |
|       |--- Detect Congested Links
|       |   |
|       |   |--- Check each link's capacity.
|       |       |
|       |       |--- Is the link congested?
|       |           |
|       |           |--- Yes
|       |           |   |
|       |           |   |--- Choose alternative link.
|       |           |
|       |           |--- No
|       |
|       |--- Handle Congestion Event
|           |
|           |--- Find the congested link.
|           |
|           |--- Choose an alternative link.
|           |
|           |--- Update Routing Table
|               |
|               |--- Update link used capacities.
|               |
|               |--- Update routing on routers
|               |
|               |--- Display updated link capacities.
|
|--- Receive Messages via Flask
    |
    |--- Congestion Alert Received
        |
        |--- Call Handle Congestion Event
        |   |
        |   |--- Handle congestion event logic
        |
        |--- Return "Message received”.
|
|--- Shutdown En-SDWANController
|
End
```

Figure 40: Simple illustration of the En-SD-WAN Controller Algorithms

5.8.5: Mathematical function for En-SDWAN Controller:

A simple mathematical function or modelling is used in this research to describe the python programming codes for the En-SDWAN controller. The mathematical model for the En-SDWAN controller is shown below in Figure 41.

Variables:

- ***link_capacities:*** *A dictionary mapping link IDs to link capacities.*
- ***routing_table****: A dictionary mapping router IDs to routing tables.*
- ***congested_links****: A set of link IDs that are currently congested.*

Functions*:*

- ***initialize_routing_tables():*** *This function initializes the routing tables on all routers. It sets the default WANs capacity and sets up the initial routing on the routers.*
- ***detect_congested_links():*** *This function checks the capacity of each link to determine if it is congested. A link is considered congested if its capacity is exceeded.*
- ***handle_congestion_event(congested_link):*** *This function takes a congested link as input and performs the following steps:*
  - *Finds an alternative link to use.*
  - *Updates the routing table to use the alternative link.*
  - *Updates the link used capacities.*
  - *Displays the updated link capacities.*
- ***receive_messages():*** *This function receives messages via Flask. If a congestion alert message is received, it calls the handle_congestion_event() function to handle the congestion event.*

Model:

The En-SDWANController process works as follows:

1. *It initializes the routing tables on all routers.*
2. *It enters a loop where it continuously detects congested links and handles congestion events.*
3. *To detect congested links, it checks the capacity of each link. If a link's capacity is exceeded, the link is considered congested.*
4. *To handle a congestion event, the En-SDWANController finds an alternative link to use and updates the routing table to use the alternative link. It also updates the link used capacities and displays the updated link capacities.*
5. *The En-SDWANController also receives messages via Flask. If a congestion alert message is received, it calls the handle_congestion_event() function to handle the congestion event.*

The En-SDWANController process can be modeled using the following states:

- *Initialize: In this state, the En-SDWANController initializes the routing tables on all routers.*
- *Detect Congestion: In this state, the En-SDWANController detects congested links.*
- *Handle Congestion: In this state, the En-SDWANController handles congestion events.*
- *Receive Messages: In this state, the En-SDWANController receives messages via Flask.*

The ***En-SDWANController*** *transitions between states as follows:*

- *From Initialize to Detect Congestion: After initializing the routing tables, the En-SDWANController transitions to the Detect Congestion state.*
- *From Detect Congestion to Handle Congestion: If a congested link is detected, the En-SDWANController transitions to the Handle Congestion state.*
- *From Handle Congestion to Detect Congestion: After handling a congestion event, the En-SDWANController transitions to the Detect Congestion state.*
- *From Receive Messages to Handle Congestion: If a congestion alert message is received, the En-SDWANController transitions to the Handle Congestion state.*
- *From Receive Messages to Detect Congestion: If a non-congestion alert message is received, the En-SDWANController transitions to the Detect Congestion state.*

The SDWANController process terminates when it receives a shutdown message.

Figure 41: Mathematical Function for En-SD-WAN Controller Program

5.9: Justification

In this section, a detailed examination of the rationale behind the utilisation of specific components and devices, as well as the emulation methods employed in this research is justified. Initially, it is the adoption of GNS3, a network emulator chosen for this study. GNS3 was selected due to its open-source nature, which permits programmable features within the devices. It stands as one of the standard emulators employed in both research and academic testbeds.

Furthermore, within the SD-WAN architecture, the routing process incorporates the use of static routes. This choice to employ static routes is intended to enhance understanding of routing concepts comprehensively. The SD-WAN controller oversees routes within the SD-WAN architecture by establishing and modifying a set of statically assigned routes within the edge routers.

Also, the Python programming tool is used over languages such as Java, C, and C++ in the execution of this research. In contemporary network architecture, Python emerges as a user-friendly programming tool compared to traditional languages. This is because of the challenges associated with implementing Java, C, and C++ within the Docker container using GNS3. Python offers simplicity in writing, implementation, incorporation, and features specific modules like "Pyshark" that can be seamlessly integrated throughout the design process.

Additionally, is the incorporation of the assumptions and estimations regarding the bandwidth of each WAN represented in our research design. In a typical network, bandwidth fluctuates based on various factors, and SD-WAN encompasses multiple WAN links with varying bandwidth requirements, owing to the diverse nature of transported applications. These assigned values are determined through a thorough evaluation process, encompassing industry standards for cable connections, and based on previous experiences of service level agreements at various points. Also, our WANs are assigned varying bandwidths, as elaborated in Table 12, to accommodate these diverse features. Furthermore, the python code developed is verified and interpreted in a simple expression using mathematical functions or models.

5.10: Conclusion

This chapter provides an overview of the En-SDWAN architecture and its realisation through Python programming tools. The initial sections involve an analysis of pivotal components utilised in creating the network architecture. Subsequently, a comprehensive examination of the traffic monitor is presented, encompassing requisites, design specifics, packet classification, and network configurations. A similar in-depth analysis is conducted for the En-

SD-WAN controller, explaining the rationale behind the selection of components and tools employed throughout the implementation process. The subsequent chapter will focus into a detailed account of the outcomes derived from implementing this En- SDWAN architecture.

CHAPTER SIX: RESULTS AND DISCUSSION

This chapter aims to investigate the key performance metrics to evaluate and discuss the results of the design implemented in the previous chapter, aligning with the research objectives.

6.1: Overview

Evaluating the performance metrics of the SD-WAN network design is crucial for analysing its performance in key areas, such as congestion level, packet loss, and latency. By examining these performance metrics, it helps to gain insights into the improvements made to the current SD-WAN architecture based on network performance. The presentation of results will focus on the functional results obtained from the new logical architecture of SD-WAN using python generated traffic flows and real time traffic flows using File Transfer Protocol based on different network scenario. The results obtained will be compared to when the traffic monitoring function and the controller logic is deactivated in the Enhanced SD-WAN. This will enable comparison of the performance of the two models to measure any improvement in the proposed En-SD-WAN architecture.

Also, there is a range of tools available for measuring performance in an SD-WAN architecture, including commercial products like Spirent TestCenter, Ixia, Netscout, and Obkio. Some of the tools are open-source and command-line tools such as Wireshark, Iperf, mtr, and ping, which are also considered for performance measurement. Wireshark, Iperf and ping are chosen because they seamlessly integrate into the SD-WAN architecture designed in the GNS3 environment.

6.2: Congestion Control Algorithm Result

This section focused on the presentation of Congestion Control Algorithm result which is obtained in the analysis of the RENO congestion. These results explained the performance of the Reno congestion control in terms of throughput and the effect of changes in windows size in the end-to-end performance operation. Figures 42 and 43 are derived from a simulation of TCP traffic flows employing the RENO technique and originating from two FTP sessions. These Figures (Figure 42 and 43) show the resulting throughput and window size respectively.

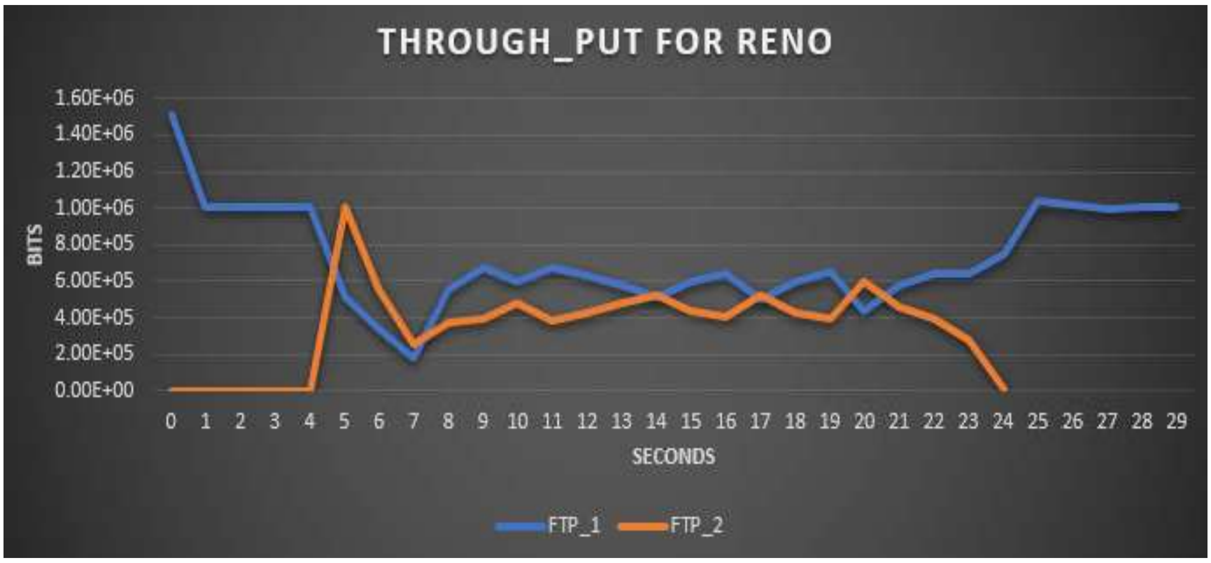


Figure 42:Throughput result for Reno Congestion Control Algorithm

Figure 42: shows the competitive nature of the flow packets in the transmission line with respect to time for the two applications using the RENO Congestion Control algorithms. The results show that the throughput starts to decrease at around 4 seconds. This is because the network is becoming congested, and the sender is having to slow down the transmission of packets. The throughput continues to fluctuate until it reaches a minimum value of around 10000 packets per second at around 25 seconds. The throughput then starts to increase again at around 7 seconds. This is because the congestion in the network has started to reduce, and the client is able to transmit more packets. This indicates that the network is no longer congested, and the sender can transmit packets at full speed. When a packet loss occurs, TCP Reno reduces the window size to half of its previous value. This helps to reduce the amount of congestion in the network.

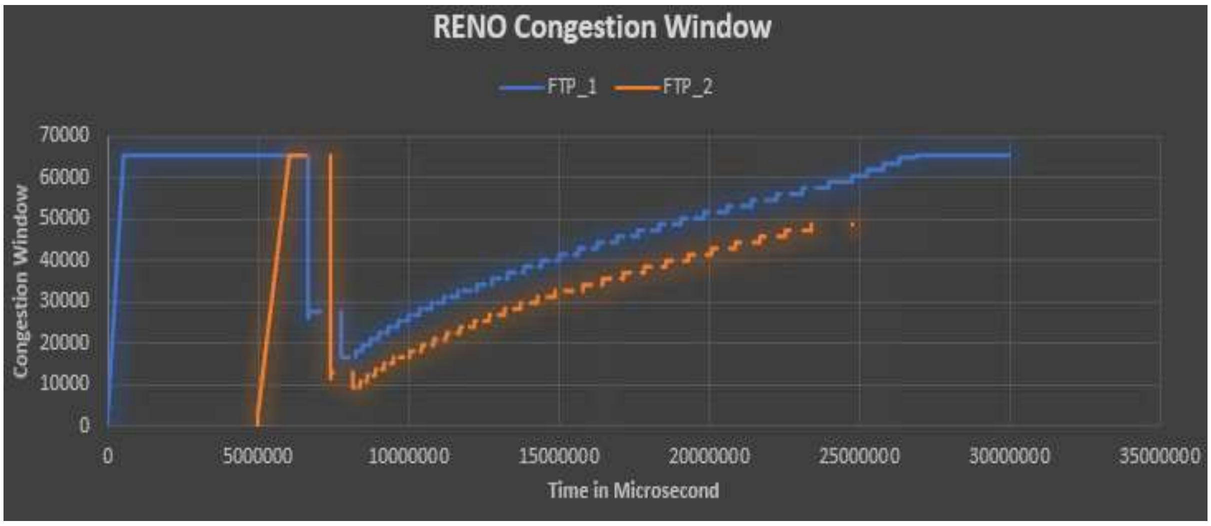


Figure 43: Throughput result for Reno window size

Figure 43 which shows the window size with respect to time. The flow starts in the slow start phase, where the window size increases exponentially. The window size reaches its maximum value at around 20000000 microseconds. In the congestion avoidance phase, the window size increases by one Maximum Segment Size (MSS) for each acknowledgement received. The results show that the window size starts to decrease at around 65000000 microseconds. This is because the flow has experienced a packet loss. When a packet loss occurs, the TCP sender reduces the window size to half of its previous value. The flow then enters the fast recovery phase. In the fast recovery phase, the window size increases by one MSS for each duplicate acknowledgement received. The result shows that the window size starts to increase, and this is because the flow has received three duplicate acknowledgements, which indicates that the lost packet has been retransmitted and received by the receiver. Examining the results in Figure 43, it shows the slow start of the window size to the point of competition for sending of data in the network. In analysing both results, the changes in the behaviour of the window size identifies congestion in the network.

6.3: SD-WAN Testing (Traffic Pattern)

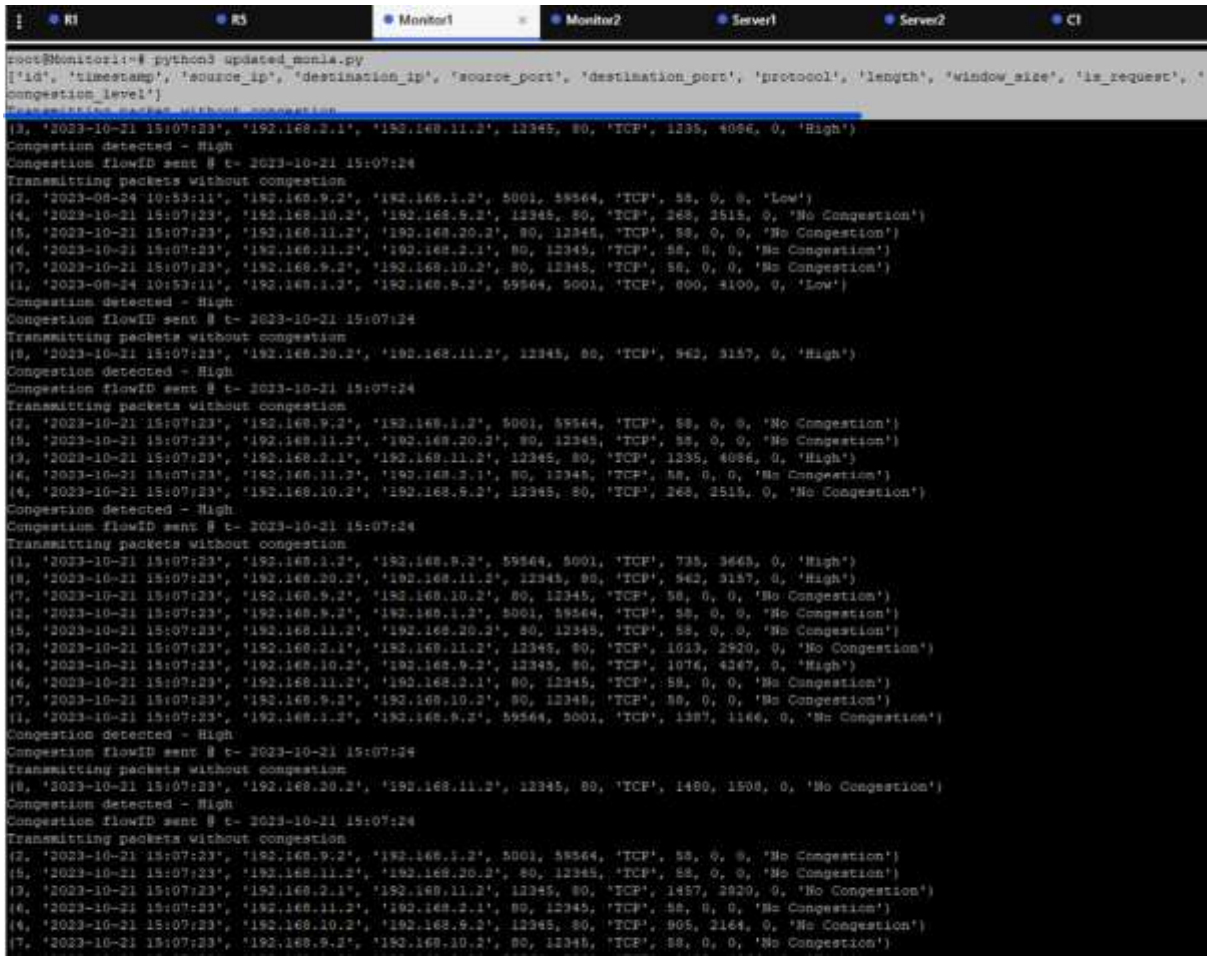
In examining the functional performance of the new logical architecture implemented, it becomes imperative to analyse the diverse traffic patterns and types, thereby facilitating an evaluation of SD-WAN performance. In the GNS3 architecture, the used traffic pattern emulates real-world traffic scenarios, mirroring the actual flow within a network. This practice holds significant relevance in ensuring the reliability of the Enhanced SD-WAN model when deployed in a physical network architecture. Hence, within the scope of this research, the investigation of file transfer based on python generated packets and real time packets been transmitted from client to serve is analysed. The file transfer pattern entails the transmission of text files from clients to servers, with varying file sizes and enhancing effective communication protocols for the purpose of analysing the performance.

6.4: Functional Result (SD-WAN Testing)

The functional results show the capability of the Enhanced SD-WAN architecture implemented in this research. The functional result comprises the bandwidth and congestion working principle. These results are screenshot of the working operation of the Enhanced SD-WAN in terms of bandwidth usage and congestion plus a general demonstration of the entire system in terms of the working operation. Further, results in terms of working operation of the traffic monitor and Enhanced SD-WAN controller are referenced in the Appendix.

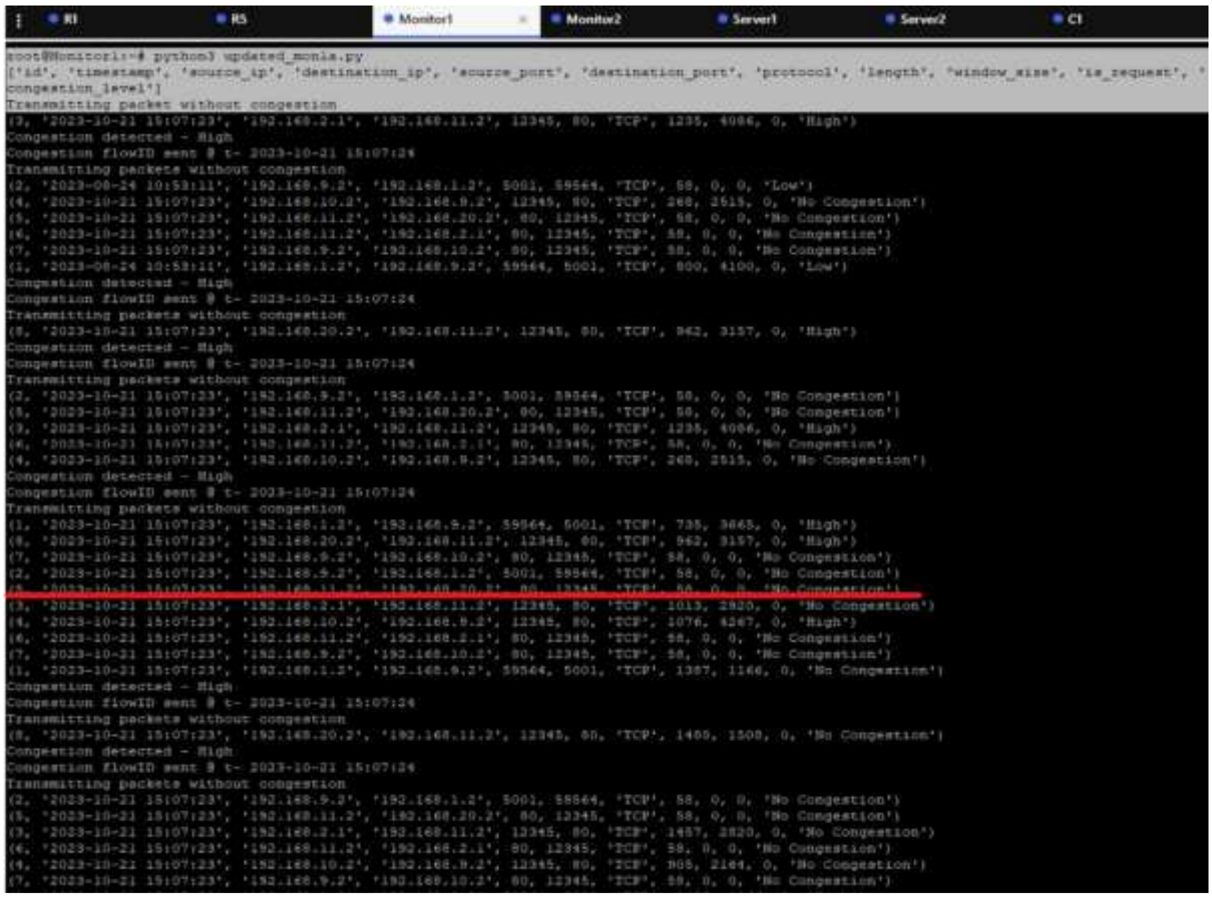
6.4.1: Bandwidth and Congestion Result Presentation

One of the objectives of this research is to improve the load balancing of the SD-WAN architecture such that it can respond in real time to the dynamic behaviour of each traffic flow and respond to any given traffic flow which has experienced congestion. To ensure the implementation of this procedure, Figure 44 shows the current route of flow ID 3 which indicate packet congestion from client IP address source 192.168.2.1 to server destination address of 192.168.11.2. When this flow ID link is congested, there will be a route change which changes the route from the current one to another route based on the available bandwidth computation. In the successful implementation of the route, it will lead to this link not being congested. The blue line indicates when the link was congested, and the red line indicates when the link is no longer in congestion because of the implementation of the routing update. Figure 44 shows the congested link in the traffic monitor and Figure 45 shows the uncongested link when the routing updates have been achieved successfully.



```
root@Monitor1:~# python3 updated_monla.py
['id', 'timestamp', 'source_ip', 'destination_ip', 'source_port', 'destination_port', 'protocol', 'length', 'window_size', 'is_request', '
congestion_level']
Transmitting packets without congestion
(3, '2023-10-21 15:07:23', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1235, 4096, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(2, '2023-08-24 10:53:11', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'Low')
(4, '2023-10-21 15:07:23', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 268, 2515, 0, 'No Congestion')
(5, '2023-10-21 15:07:23', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(6, '2023-10-21 15:07:23', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(7, '2023-10-21 15:07:23', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-08-24 10:53:11', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 800, 3100, 0, 'Low')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(8, '2023-10-21 15:07:23', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 962, 3157, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(2, '2023-10-21 15:07:23', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 15:07:23', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 15:07:23', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1235, 4096, 0, 'High')
(6, '2023-10-21 15:07:23', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(4, '2023-10-21 15:07:23', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 268, 2515, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(1, '2023-10-21 15:07:23', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 735, 3665, 0, 'High')
(8, '2023-10-21 15:07:23', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 962, 3157, 0, 'High')
(7, '2023-10-21 15:07:23', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(2, '2023-10-21 15:07:23', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 15:07:23', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 15:07:23', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1023, 2920, 0, 'No Congestion')
(4, '2023-10-21 15:07:23', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1076, 4267, 0, 'High')
(6, '2023-10-21 15:07:23', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(7, '2023-10-21 15:07:23', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 15:07:23', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 1387, 1166, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(8, '2023-10-21 15:07:23', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1480, 1508, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(2, '2023-10-21 15:07:23', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 15:07:23', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 15:07:23', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1457, 2820, 0, 'No Congestion')
(6, '2023-10-21 15:07:23', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(4, '2023-10-21 15:07:23', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 905, 2164, 0, 'No Congestion')
(7, '2023-10-21 15:07:23', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
```

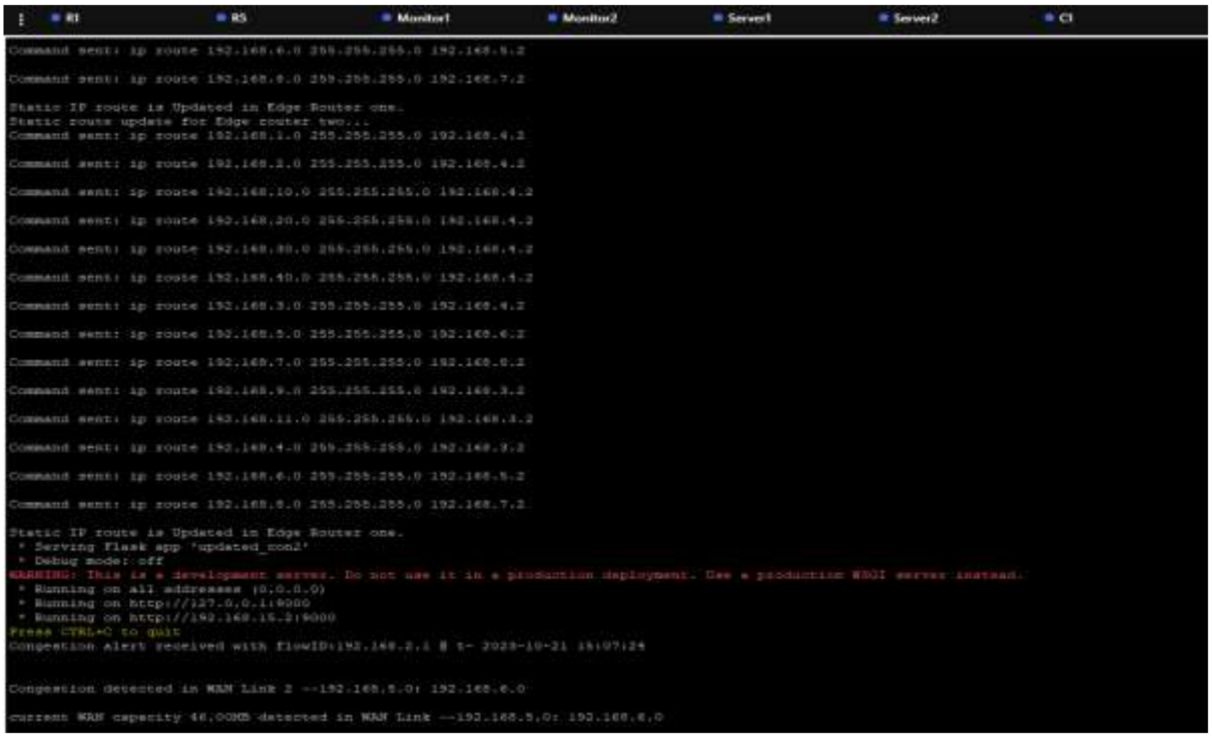
Figure 44: Functional result showing link is not congested because of re-routing.



```
root@Monitor1:~# python3 updated_monia.py
['id', 'timestamp', 'source_ip', 'destination_ip', 'source_port', 'destination_port', 'protocol', 'length', 'window_size', 'is_request', '
congestion_level']
Transmitting packet without congestion
(3, '2023-10-21 15:07:23', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1235, 4086, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(2, '2023-08-24 10:53:11', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'Low')
(4, '2023-10-21 15:07:23', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 268, 2515, 0, 'No Congestion')
(5, '2023-10-21 15:07:23', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(6, '2023-10-21 15:07:23', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(7, '2023-10-21 15:07:23', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-08-24 10:53:11', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 800, 4100, 0, 'Low')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(8, '2023-10-21 15:07:23', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 962, 3157, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(2, '2023-10-21 15:07:23', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 15:07:23', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 15:07:23', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1235, 4086, 0, 'High')
(6, '2023-10-21 15:07:23', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(4, '2023-10-21 15:07:23', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 268, 2515, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(1, '2023-10-21 15:07:23', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 735, 3665, 0, 'High')
(8, '2023-10-21 15:07:23', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 962, 3157, 0, 'High')
(7, '2023-10-21 15:07:23', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(2, '2023-10-21 15:07:23', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 15:07:23', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 15:07:23', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1013, 2920, 0, 'No Congestion')
(4, '2023-10-21 15:07:23', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1076, 4267, 0, 'High')
(6, '2023-10-21 15:07:23', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(7, '2023-10-21 15:07:23', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 15:07:23', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 1387, 1166, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(8, '2023-10-21 15:07:23', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1400, 1508, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(2, '2023-10-21 15:07:23', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 15:07:23', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 15:07:23', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1457, 2820, 0, 'No Congestion')
(6, '2023-10-21 15:07:23', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(4, '2023-10-21 15:07:23', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 905, 2164, 0, 'No Congestion')
(7, '2023-10-21 15:07:23', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
```

Figure 45: Functional result showing link is not congested because of re-routing.

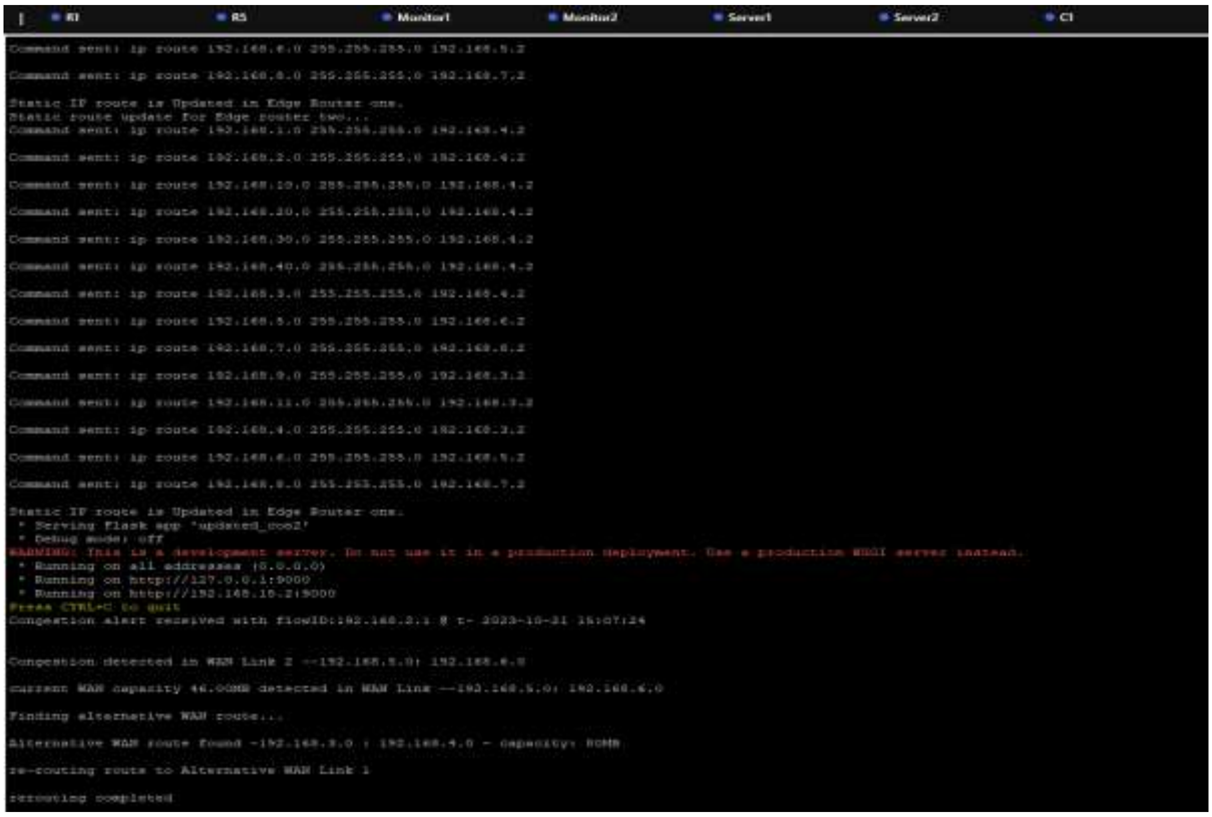
Furthermore, it is the use of bandwidth amongst the various WAN links and the way they adapt to changes in the network behaviour. The initial bandwidth is set to be 100Mbps each for the three WAN links and aggregate flows are set to be less than that value at the beginning of the packet flows. When there is a change in the network behaviour because of the network flows passing between the client and the server this will lead to changes in the bandwidth utilisation.



```
Command sent: ip route 192.168.6.0 255.255.255.0 192.168.5.2

Command sent: ip route 192.168.8.0 255.255.255.0 192.168.7.2

Static IP route is Updated in Edge Router one.
Static route update for Edge router two...
Command sent: ip route 192.168.1.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.2.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.10.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.20.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.30.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.40.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.3.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.5.0 255.255.255.0 192.168.6.2

Command sent: ip route 192.168.7.0 255.255.255.0 192.168.8.2

Command sent: ip route 192.168.9.0 255.255.255.0 192.168.3.2

Command sent: ip route 192.168.11.0 255.255.255.0 192.168.3.2

Command sent: ip route 192.168.4.0 255.255.255.0 192.168.3.2

Command sent: ip route 192.168.6.0 255.255.255.0 192.168.5.2

Command sent: ip route 192.168.8.0 255.255.255.0 192.168.7.2

Static IP route is Updated in Edge Router one.
 * Serving Flask app 'updated_con2'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:9000
 * Running on http://192.168.15.2:9000
Press CTRL+C to quit
Congestion alert received with flowID:192.168.2.1 @ t- 2023-10-21 15:07:24


Congestion detected in WAN Link 2 --192.168.5.0: 192.168.6.0

current WAN capacity 46.00MB detected in WAN Link --192.168.5.0: 192.168.6.0
```

Figure 46: Functional results for Initial Bandwidth Configuration of WANs in the En-SDWAN Controller

The initial bandwidth configuration will always change depending on the network adaptation and configuration as streams of packet passes through the various WAN networks. Figure 47 shows the changes in network bandwidth because of changes in the network flow streams between the WANs.



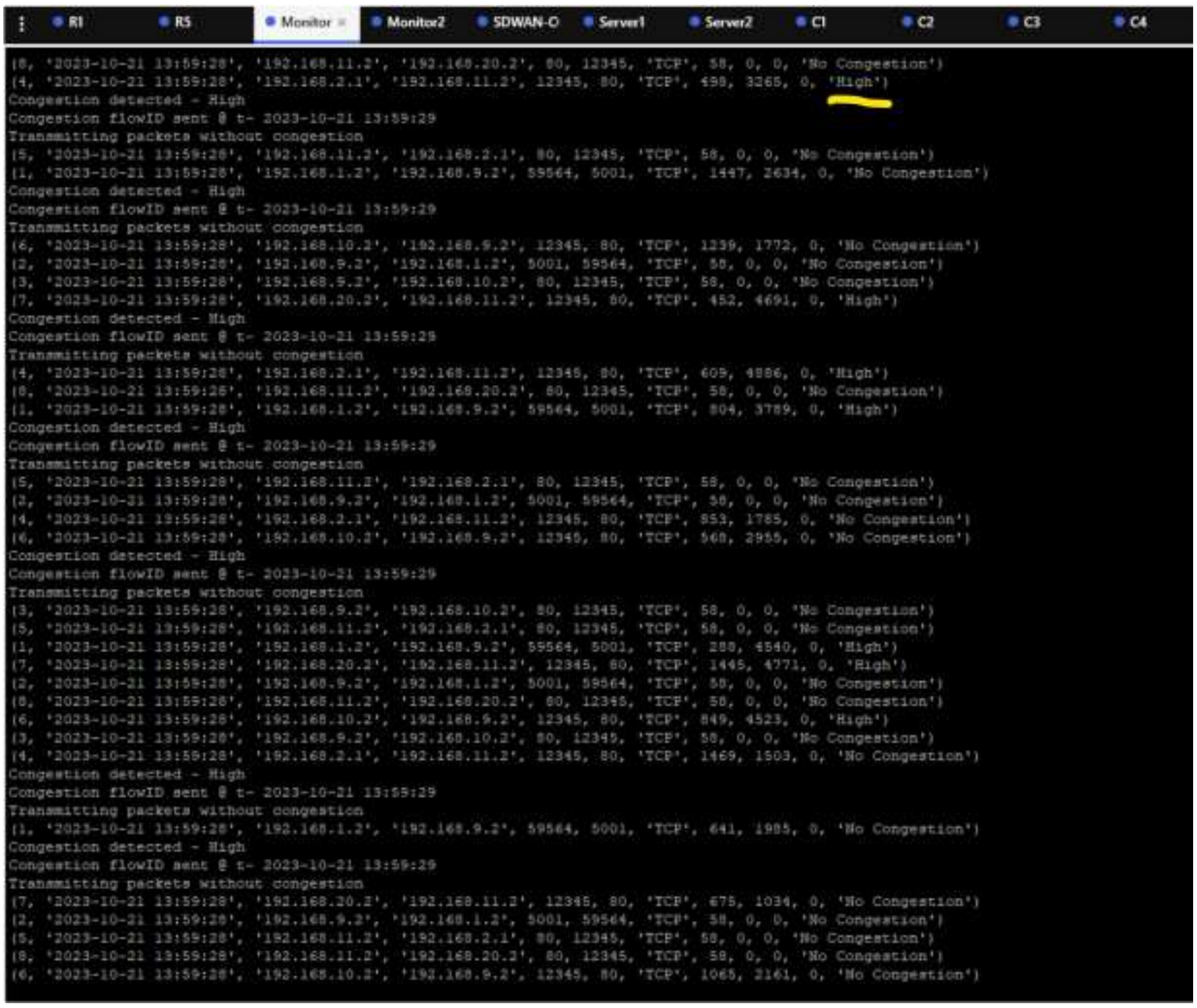
```
Command sent: ip route 192.168.6.0 255.255.255.0 192.168.5.2

Command sent: ip route 192.168.8.0 255.255.255.0 192.168.7.2

Static IP route is Updated in Edge Router one.
Static route update for Edge router two...
Command sent: ip route 192.168.1.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.2.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.10.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.20.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.30.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.40.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.3.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.5.0 255.255.255.0 192.168.6.2

Command sent: ip route 192.168.7.0 255.255.255.0 192.168.8.2

Command sent: ip route 192.168.9.0 255.255.255.0 192.168.3.2

Command sent: ip route 192.168.11.0 255.255.255.0 192.168.3.2

Command sent: ip route 192.168.4.0 255.255.255.0 192.168.3.2

Command sent: ip route 192.168.6.0 255.255.255.0 192.168.5.2

Command sent: ip route 192.168.8.0 255.255.255.0 192.168.7.2

Static IP route is Updated in Edge Router one.
 * Serving Flask app 'updated_con2'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:9000
 * Running on http://192.168.15.2:9000
Press CTRL+C to quit
Congestion alert received with flowID:192.168.2.1 @ t- 2023-10-21 15:07:24


Congestion detected in WAN Link 2 --192.168.5.0: 192.168.6.0

current WAN capacity 46.00MB detected in WAN Link --192.168.5.0: 192.168.6.0

Finding alternative WAN route...

Alternative WAN route found -192.168.3.0 : 192.168.4.0 - capacity: 80MB

re-routing route to Alternative WAN Link 1

rerouting completed
```

Figure 47: Functional Results showing change in initial Bandwidth Configuration for WANs

The functional results show the working operation of the implemented new logical architecture of SD-WAN with the monitor function which focuses on the traffic management process and En-SDWAN controller. Each of the functional results have shown the working operation in terms of detecting congestion, making the network to have less bottleneck, the changes in bandwidth, monitor database and the response time of when a network congested and when it is decongested. All the various functional results have aligned with the core objectives of this research.

6.4.2: Enhanced SD-WAN Demonstration

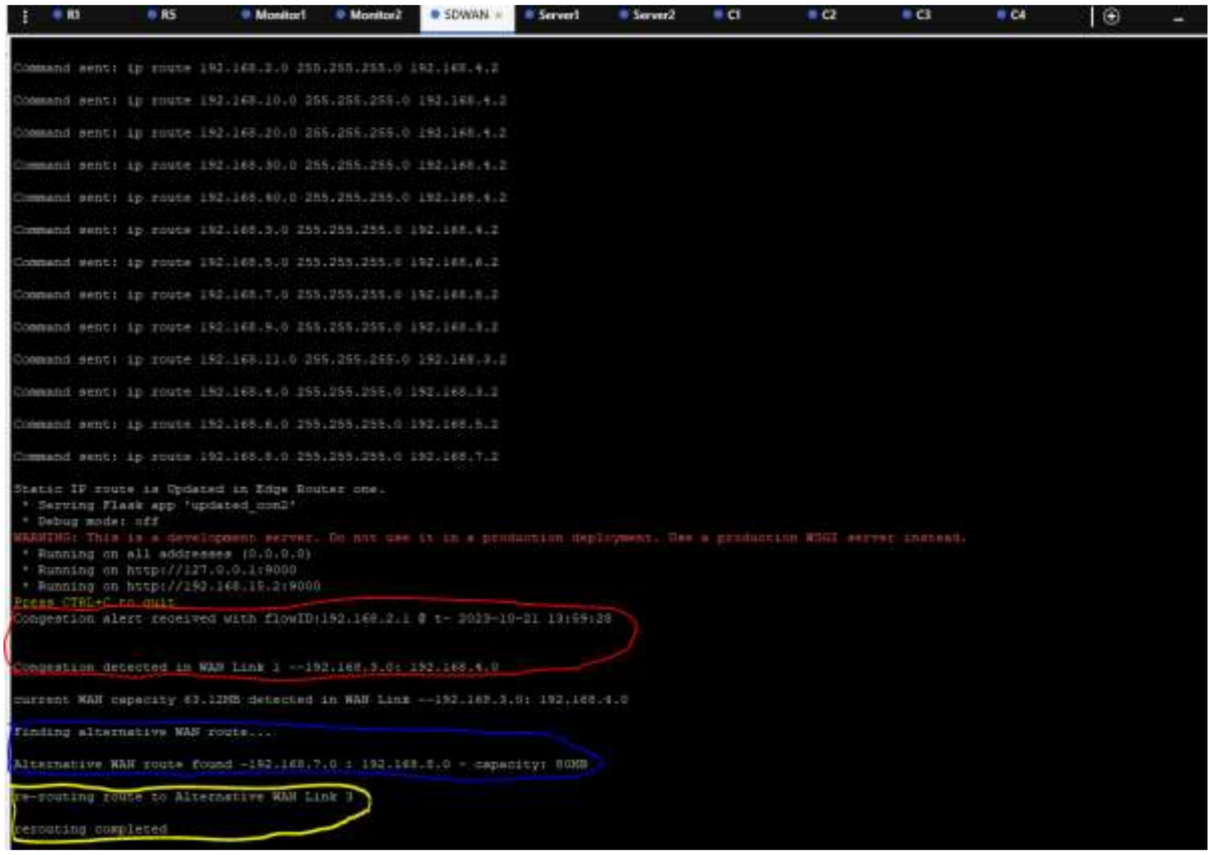
In this section a demonstration of the entire concept and implementation of the Enhanced SD-WAN system is presented. This involves detecting congestion in a particular packet flow which is caused by variations in window size, percentage of packet loss and average latency determined in the network. Figure 48 shows a packet flow which is congested, with a flow ID 4 with a source IP address of 192.168.2.1 to a destination of 192.168.11.2. This flow is congested based on the computation of the percentage of packet loss within a duration combined with latency computed for that period aligning with the variation in window size scaling of TCP. This relationship is expressed as percentage of packet loss of more than 5% of the total packets been transmitted with average latency of more than 0.5milliseconds combined with a range of window size indicating high congestion in the network. If these parameters are slightly changed to percentage of packet loss of more than 3% of the total packets being transmitted with latency of more than 0.3milliseconds combined with a range of lower window size indicate medium congestion in the network. In a condition where these parameters are not fulfilled, it means that there is no congestion in the network and for the Figure 48 the level of congestion is high. This means that it satisfies the high congestion parameters as shown in Figure 48. The congested packet is marked yellow which shows that particular Flow ID is congested.



```
(8, '2023-10-21 13:59:28', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(4, '2023-10-21 13:59:28', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 498, 3265, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(5, '2023-10-21 13:59:28', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 13:59:28', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 1447, 2634, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(6, '2023-10-21 13:59:28', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1239, 1772, 0, 'No Congestion')
(2, '2023-10-21 13:59:28', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 13:59:28', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(7, '2023-10-21 13:59:28', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 452, 4691, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(4, '2023-10-21 13:59:28', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 609, 4886, 0, 'High')
(8, '2023-10-21 13:59:28', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 13:59:28', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 804, 3789, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(5, '2023-10-21 13:59:28', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(2, '2023-10-21 13:59:28', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(4, '2023-10-21 13:59:28', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 853, 1785, 0, 'No Congestion')
(6, '2023-10-21 13:59:28', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 568, 2955, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(3, '2023-10-21 13:59:28', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 13:59:28', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 13:59:28', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 288, 4540, 0, 'High')
(7, '2023-10-21 13:59:28', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1445, 4771, 0, 'High')
(2, '2023-10-21 13:59:28', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(8, '2023-10-21 13:59:28', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(6, '2023-10-21 13:59:28', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 849, 4523, 0, 'High')
(3, '2023-10-21 13:59:28', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(4, '2023-10-21 13:59:28', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1469, 1503, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(1, '2023-10-21 13:59:28', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 641, 1985, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(7, '2023-10-21 13:59:28', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 675, 1034, 0, 'No Congestion')
(2, '2023-10-21 13:59:28', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 13:59:28', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(8, '2023-10-21 13:59:28', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(6, '2023-10-21 13:59:28', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1065, 2161, 0, 'No Congestion')
```

Figure 48: illustrates a packet flow which is congested, with a flow ID 1 with a source IP address of 192.168.1.2 and destination address of 192.168.9.2.

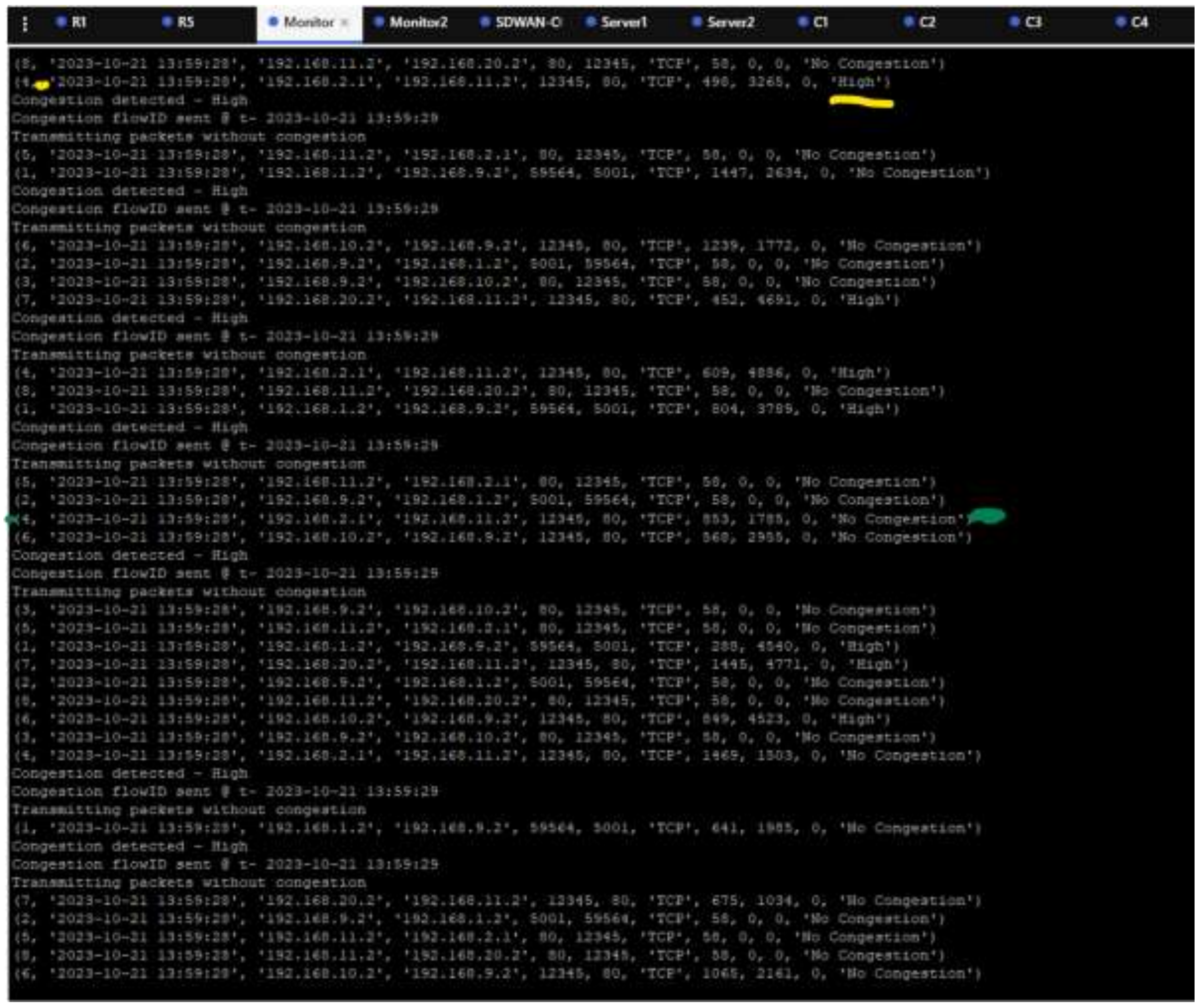
Since the flow ID 4 is detected to be in congestion and Figure 49 illustrates the congested updates received by the En-SDWAN controller. The En-SDWAN controller will now act by analysing the congestion and send routing updates to the edge routers by transferring from the current WAN based on the mapping to an alternative WAN that has bandwidth that could accommodate the flow ID 4. The En-SDWAN controller sent route updates to the edge routers to change the route of flow ID 4 from the WAN1 (192.168.3.0: 192.168.4.0) to WAN 3 (192.168.7.0: 192.168.8.0). The updates route is displayed in same Figure 49 showing the transition from WAN 1 to WAN 3 based on the bandwidth requirement. The decision from WAN 1 to WAN 3 is based on the assumed bandwidth of the three WAN's connected in the SD-WAN architecture. A predefined bandwidth is issued to each WAN which is 100 bytes per second for the purpose of the analysis and each capacity is issued a current bandwidth. Thus, decision for alternative route is based on the link capacity, utilised capacity, and unused capacity for each WAN link. The link capacity which is set at 100 bytes per second with the aim of traffic should not exceed 80% of any of the current link's capacity. In cases where WAN link traffic volume surpasses 80% of its bandwidth in the case of WAN 1, alternative WAN links is considered. The selection of an alternative link is determined by the available bandwidth, calculated by the difference between the total link capacity and the used link capacity based on bandwidth provisions where this condition is not met another link is considered. Thus, a routing function is called to send routing updates to the edge routers to activate the route for WAN 3. The red circle describes receiving congestion from the traffic monitor function which is followed by WAN bandwidth examination to determine the best route which is described with the blue circle. The route update function is called to send routing update to edge routers to implement the change route and this is described with the yellow-coloured circle.



```
Command sent: ip route 192.168.2.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.10.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.20.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.30.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.40.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.3.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.5.0 255.255.255.0 192.168.6.2

Command sent: ip route 192.168.7.0 255.255.255.0 192.168.8.2

Command sent: ip route 192.168.9.0 255.255.255.0 192.168.3.2

Command sent: ip route 192.168.11.0 255.255.255.0 192.168.3.2

Command sent: ip route 192.168.4.0 255.255.255.0 192.168.3.2

Command sent: ip route 192.168.6.0 255.255.255.0 192.168.5.2

Command sent: ip route 192.168.8.0 255.255.255.0 192.168.7.2

Static IP route is Updated in Edge Router one.
 * Serving Flask app 'updated_con2'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:9000
 * Running on http://192.168.15.2:9000
Press CTRL+C to quit
Congestion alert received with flowID:192.168.2.1 @ t- 2023-10-21 13:59:28


Congestion detected in WAN Link 1 --192.168.3.0: 192.168.4.0

current WAN capacity 43.12MB detected in WAN Link --192.168.3.0: 192.168.4.0

Finding alternative WAN route...

Alternative WAN route found -192.168.7.0 : 192.168.8.0 - capacity: 80MB

re-routing route to Alternative WAN Link 3

rerouting completed
```

Figure 49: illustrates a change of route of a congested packets by examining an alternative route.

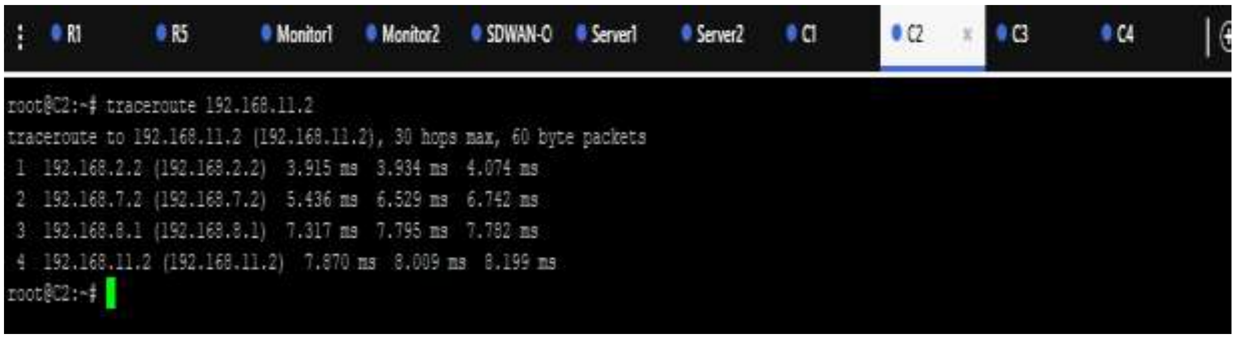
Based on the change in the route of this flow to ease the level of congestion from high to no congestion. Figure 50 shows the changes in the level of congestion status of the flow ID 4 from high to no congestion. It is important to note that figure 50 is obtained from the traffic monitor, the yellow mark indicates the congested flow ID 4, and the green mark indicates the packet is no longer congested.



```
(8, '2023-10-21 13:59:28', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(4, '2023-10-21 13:59:28', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 498, 3265, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(5, '2023-10-21 13:59:28', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 13:59:28', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 1447, 2634, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(6, '2023-10-21 13:59:28', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1239, 1772, 0, 'No Congestion')
(2, '2023-10-21 13:59:28', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 13:59:28', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(7, '2023-10-21 13:59:28', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 452, 4691, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(4, '2023-10-21 13:59:28', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 609, 4886, 0, 'High')
(8, '2023-10-21 13:59:28', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 13:59:28', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 804, 3789, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(5, '2023-10-21 13:59:28', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(2, '2023-10-21 13:59:28', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(4, '2023-10-21 13:59:28', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 853, 1785, 0, 'No Congestion')
(6, '2023-10-21 13:59:28', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 868, 2955, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:55:29
Transmitting packets without congestion
(3, '2023-10-21 13:59:28', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 13:59:28', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 13:59:28', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 288, 4640, 0, 'High')
(7, '2023-10-21 13:59:28', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1445, 4771, 0, 'High')
(2, '2023-10-21 13:59:28', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(8, '2023-10-21 13:59:28', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(6, '2023-10-21 13:59:28', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 849, 4523, 0, 'High')
(3, '2023-10-21 13:59:28', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(4, '2023-10-21 13:59:28', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1469, 1503, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(1, '2023-10-21 13:59:28', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 641, 1985, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 13:59:29
Transmitting packets without congestion
(7, '2023-10-21 13:59:28', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 675, 1034, 0, 'No Congestion')
(2, '2023-10-21 13:59:28', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 13:59:28', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(8, '2023-10-21 13:59:28', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(6, '2023-10-21 13:59:28', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1065, 2161, 0, 'No Congestion')
```

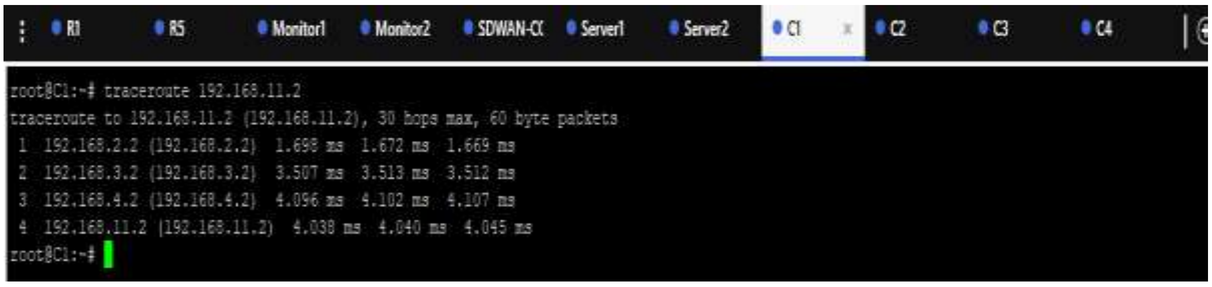
Figure 50: Functional results illustrating packet in and out of congestion.

This can be further described as well using the routing table functions in each of the packets sent from the various clients. Based on the mapping client 2 is the packet that is congested which is designated to be at WAN 1 and is moved to WAN 3. This means the traceroute for the congested have changed in terms of his route in getting to his destination. Figure 51 and 52 describes the route change for the congested flow. Recall, based on our network design WAN 1 network route is 192.168.3.0 to 192.168.4.0 and WAN 3 network route is 192.168.7.0 to 192.168.8.0. Hence, Figure 51 shows the initial route taken when the packet is being forwarded which shows the packet is congested. The traceroute Cisco command is used to show all the hops and the route taken, it is clearly shown the route or path taken is (192.168.2.2 → 192.168.7.2 → 192.168.8.1 → 192.168.11.2). When the En-SDWAN controller have issued updates for change of route and when the traceroute Cisco command is used to check, then the route path is changed to (192.168.2.2 → 192.168.3.2 → 192.168.4.2 → 192.168.11.2).



```
root@C2:~# traceroute 192.168.11.2
traceroute to 192.168.11.2 (192.168.11.2), 30 hops max, 60 byte packets
 1  192.168.2.2 (192.168.2.2)  3.915 ms  3.934 ms  4.074 ms
 2  192.168.7.2 (192.168.7.2)  5.436 ms  6.529 ms  6.742 ms
 3  192.168.8.1 (192.168.8.1)  7.317 ms  7.795 ms  7.782 ms
 4  192.168.11.2 (192.168.11.2)  7.870 ms  8.009 ms  8.199 ms
root@C2:~#
```

Figure 51: traceroute for a congested flow for clients 2.



```
root@C1:~# traceroute 192.168.11.2
traceroute to 192.168.11.2 (192.168.11.2), 30 hops max, 60 byte packets
 1  192.168.2.2 (192.168.2.2)  1.698 ms  1.672 ms  1.669 ms
 2  192.168.3.2 (192.168.3.2)  3.507 ms  3.513 ms  3.512 ms
 3  192.168.4.2 (192.168.4.2)  4.096 ms  4.102 ms  4.107 ms
 4  192.168.11.2 (192.168.11.2)  4.038 ms  4.040 ms  4.045 ms
root@C1:~#
```

Figure 52: traceroute for a non- congested flow for clients 2.

6.5: Performance Result (SD-WAN Testing)

This section focusses on analysis of the performance comparison between the Enhanced SD-WAN architecture and the traditional SD-WAN controller. In this implementation, the Enhanced SD-WAN, as depicted in Figure 27 of the design, integrates a traffic monitoring process. This process facilitates the transmission of route updates to the Enhanced SD-WAN controller, enabling dynamic routing functionality. Meanwhile, the traditional SD-WAN operates without incorporating the traffic monitoring management feature, which would otherwise send route updates to the Enhanced SD-WAN controller. In the traditional SD-WAN controller setup simulated in GNS3, the architecture features a centralized controller responsible for network management and orchestration. This controller interacts with edge devices (routers) within the network design, providing instructions for traffic routing, policy enforcement, and network optimisation without the traffic monitoring process. Basing on the working operation, the Traditional SD-WAN controller gathers information on network topology and dynamically adjusts traffic flows based on predetermined policies and conditions. The results are based on packets files transferred via the GNS3 application and packets files generated using Python script.

6.5.1: Python Script Traffic

In reference to the network design in Figure 27, there are six clients connected to two servers, as illustrated in the network diagram. Data traffic has been effectively transmitted from the various clients to the servers using packet files generated by a Python script. The primary objective of this implementation is to evaluate throughput over time during the packet transmission process. It is noteworthy that in the GNS3 design implementation, the GNS3 interfaces are depicted as Gigabytes. For the purposes of this implementation, the data text file transferred from client one to client six is specified as follows: 100KB, 150KB, 200KB, 250KB, 300KB, and 350KB, respectively, within a duration of 10 seconds. Standard parameters for MTU values, speed, and bandwidth are utilised, set at 1500 bytes and 1 Gbps for speed and bandwidth, respectively. Furthermore, it is essential to mention that the text files of these packet files are created as (.txt) files, with information written in the text file format. Figure 53 displays the throughput results over time for the Enhanced and Traditional SD-WAN architecture. According to the findings in Figure 53 with reference to the Enhanced SD-WAN, the highest bitrate recorded is 22.28Mbps, while the lowest is measured at 17.61Mbps. The maximum throughput observed reflects efficient data file transmission because of the amount of data been transferred from the clients to the servers although there is a slight fluctuation in subsequent time intervals.

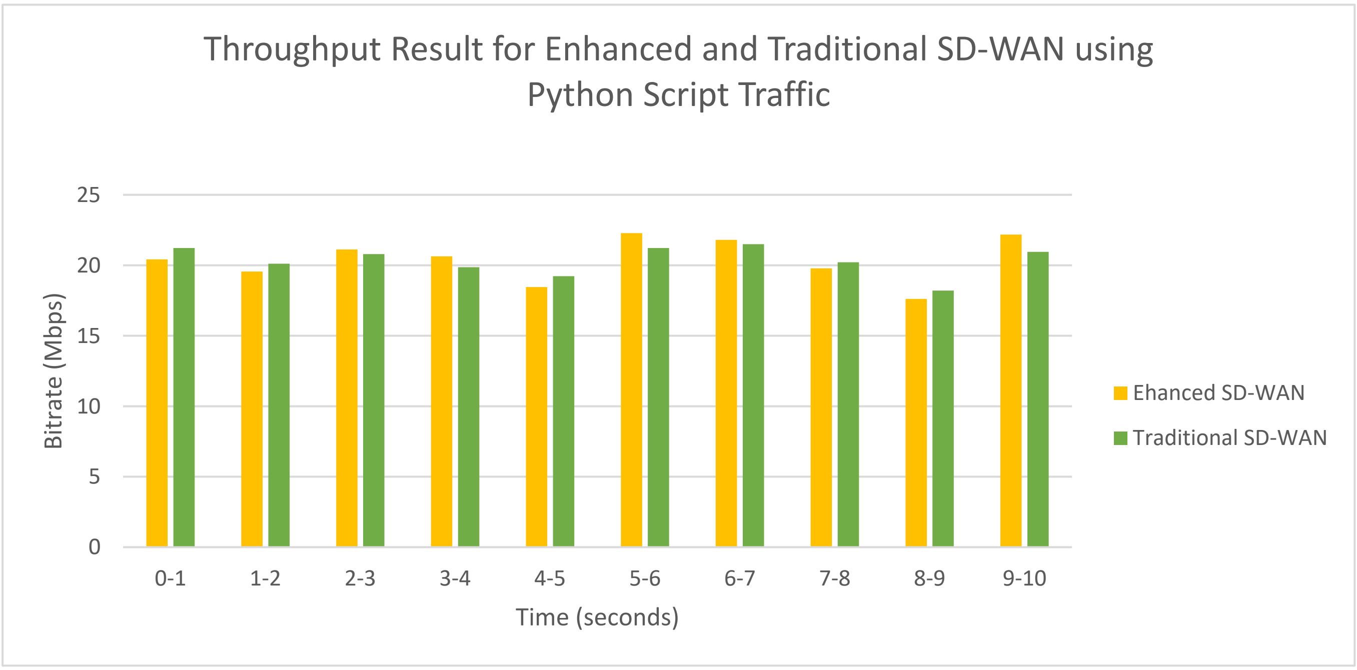


Figure 53: Throughput result for Enhanced and Traditional SD-WAN using Python Script Traffic

Also, it is the implementation of the same network conditions without the Enhanced capability and same data files has been transmitted to observe the behaviour and performance. Figure 53 shows the results of bitrate with respect to time in describing the throughput of the end-to-end operation. Based on the results obtained, it is understood there is a consistency in bitrate transmitted with respect to time compared to the Enhanced SD-WAN. This is because there is no congestion experienced because of the amount of data transferred between the various clients and the servers. Furthermore, in comparison the outcomes derived from both the traditional SD-WAN and Enhanced SD-WAN models to assess the operational and performance differences. The analysis reveals that the Enhanced SD-WAN performs in similar capacity with the traditional SD-WAN across various parameters, including maximum, average, and minimum bitrate, as illustrated in Figure 54 using the Python Scripts. The consistent performance for the Enhanced SD-WAN and traditional SD-WAN is because of the lack of congestion which is based on the data size, given bandwidth and network speed transmitted between the various clients and servers.

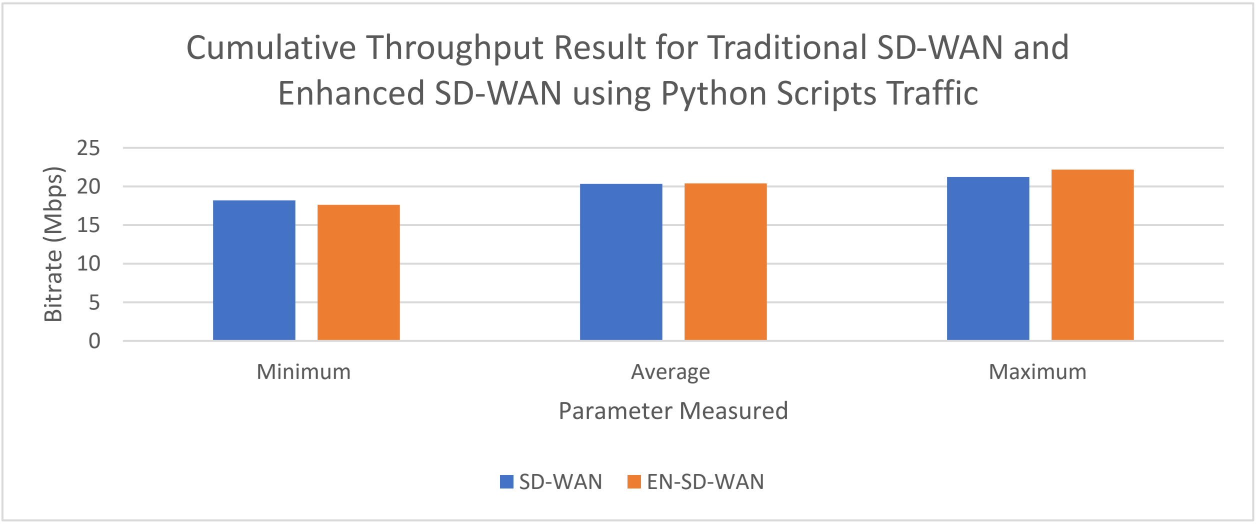


Figure 54: Cumulative Throughput result for Traditional SD-WAN and Enhanced SD-WAN in using Python Scripts Traffic.

6.5.2: GNS3 Application Traffic

This section explores the transferring of data from the various clients to the servers using traffic generated via the GNS3 application. In this case, the GNS3 applications send same text files with same amount of data size between the various clients to server. This size of the text files is 100KB, 150KB, 200KB, 250KB, 300KB, and 350KB, respectively for the various clients to the server, within a duration of 10 seconds. Standard parameters for MTU values, speed, and bandwidth are utilised, set at 1500 bytes and 1 Gbps for speed and bandwidth, respectively. Focusing on the use of the Enhanced SD-WAN design, the results are consistent across the various time duration as the highest throughput rate is lower than 35Mbps, depicted in Figure 55.

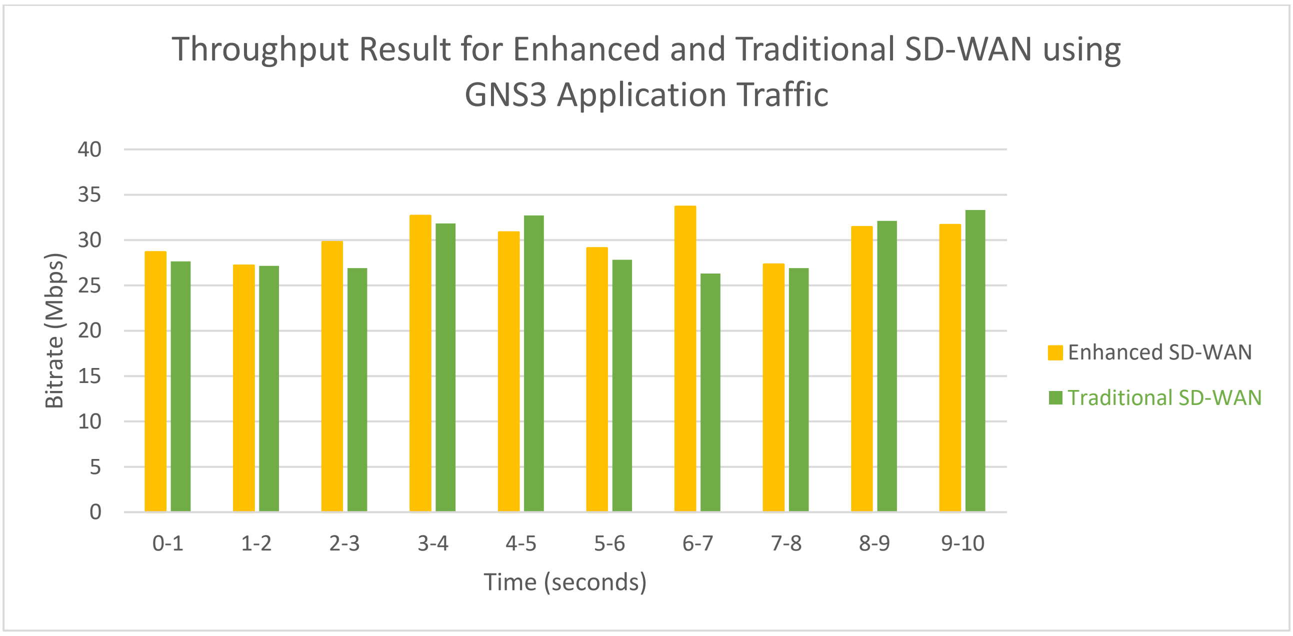


Figure 55: Throughput result for Enhanced and Traditional SD-WAN using GNS3 Application Traffic.

While both results of the Enhanced SD-WAN controller and Traditional Controller exhibits a similar pattern with a notable high throughput indication between 6 to 7 seconds, with a lower throughput in the 1 to 2 seconds timeframe. The figure 55 indicate the behaviour and performance of the Traditional SD-WAN using the GNS3 application traffic.

The comparison of the Enhanced SD-WAN and Traditional SD-WAN using the traffic condition shows consistent results because of the little or no effect of protocol overhead or congestion in the network. This is the expected outcome for both the Enhanced and traditional SD-WAN controller in terms of performance when the network is not experiencing any form of congestion in the network. However, the Enhanced SD-WAN controller have a slight edge (very minimal) better performance above the traditional SD-WAN controller as shown in Figure 56. Figure 56 results describes that in the maximum, minimum and average throughput in the implementation of Enhanced SD-WAN and traditional SD-WAN throughput.

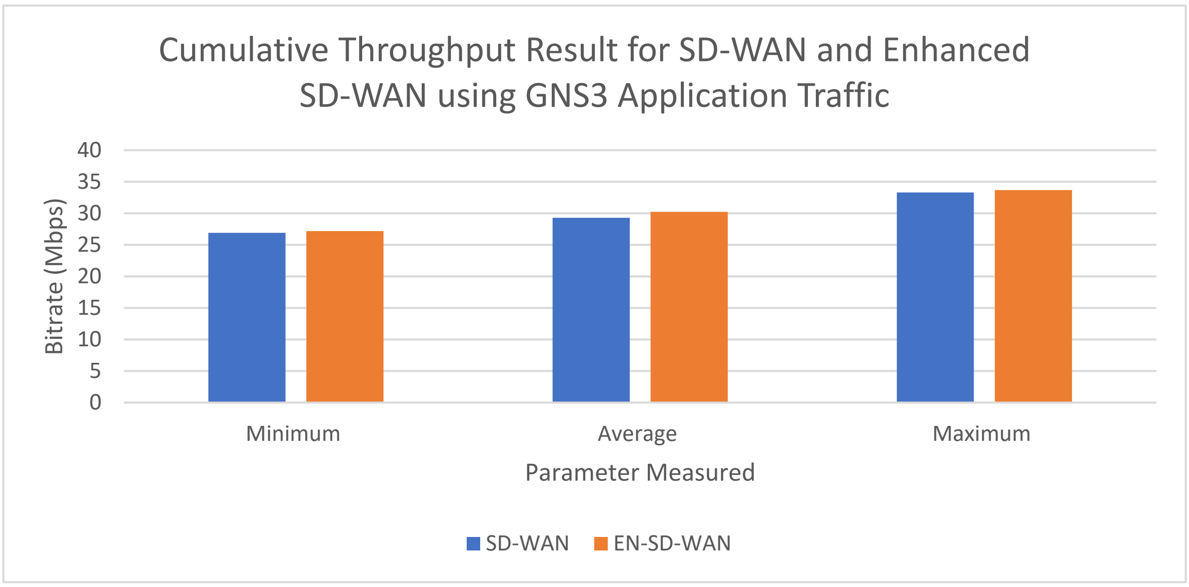


Figure 56: Cumulative Throughput result for Traditional SD-WAN and Enhanced SD-WAN using GNS3 Application Traffic.

6.5.3: Link Failure Traffic Using Python Script

This section focused on link failure in one of the WAN links to discuss how the network converge and process load sharing amongst the various WANs. In reference to Figure 27, which is the network design, there are three WANs connected between the various clients to the servers. In implementing the link failure scenario, one of the links which is the middle WAN link is suspended leading to failure of flow of traffic between the various clients to the servers. The traffic will be automatically redistributed among the remaining WAN links based on the bandwidth availability of those WANs link. This scenario applies to both the Enhanced SD-WAN and Traditional SD-WAN (Non-Enhanced SD-WAN) designs under consideration considering the same traffic type, size of data and settings for speed, bandwidth and MTU values in both designs. Figure 57 shows the results of bitrate with respect to time in describing the throughput of the end-to-end operation when there is a failure in the link using Enhanced and traditional SD-WAN.

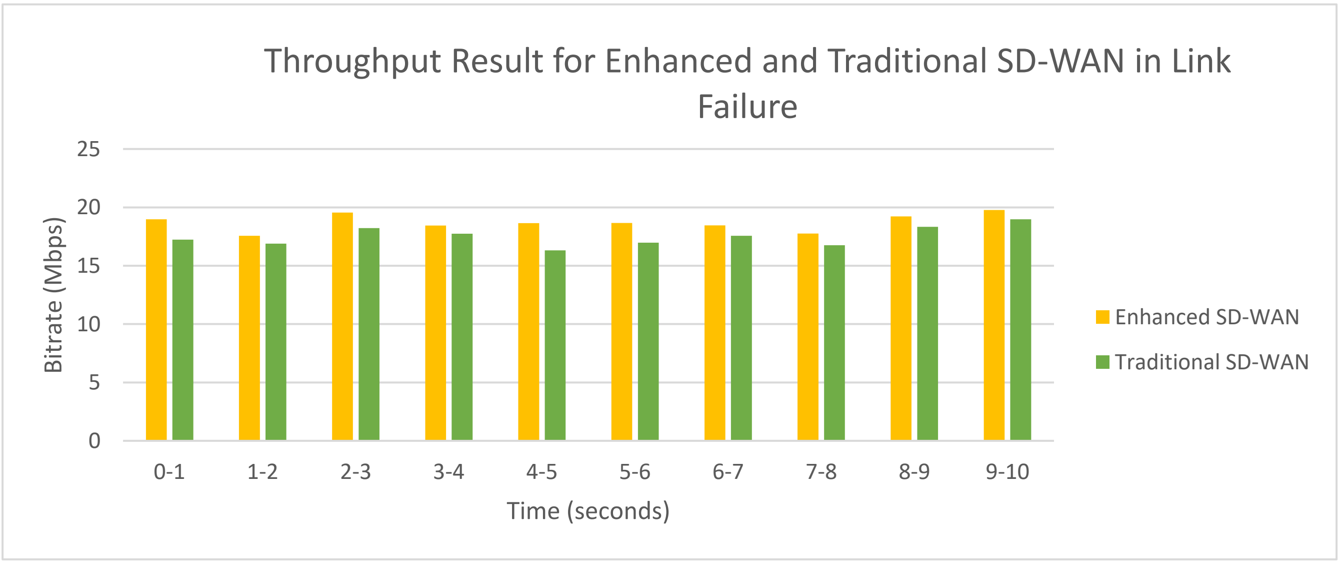


Figure 57: Throughput result for Enhanced and Traditional SD-WAN in Link Failure.

The results in the Enhanced SD-WAN which is marked as orange describe that the maximum throughput is at 9 to 10 seconds valued at 19.78Mbps and minimum is experienced at 1 to 2 seconds with a value of 17.56Mbps. Based on the same network conditions, the simulation is implemented without the traffic management system in the case of the traditional SD-WAN controller and the result is described in green shown in Figure 57. The result shows a slight drop in the throughput performance whereby the maximum throughput is 18.98Mbps at 9 to 10 seconds and minimum throughput is 16.32Mbps at 1 to 2 seconds. The drop in throughput is because of the lack of functionalities in the traffic management policy, providing information to the SD-WAN controller to make effective decisions about the network behaviour when a link fail.

The comparative analysis between Traditional SD-WAN and Enhanced SD-WAN concerning link failure, facilitated by packets generated using python scripts, reveals the slight better performance of the Enhanced SD-WAN across minimum, average, and maximum throughput metrics. This emphasises Enhanced SD-WAN's adherence to efficient traffic management policies, which effectively reroute packets along alternative paths in instances of WAN link failure. Furthermore, it indicates that network convergence occurs more rapidly within the Enhanced SD-WAN architecture compared to the traditional SD-WAN architecture following a link failure event. This underscores the significance of implementing an effective traffic management policy within the traditional SD-WAN framework capable of dynamically adapting to fluctuations in network conditions, including WAN link failures. The throughput results for traditional SD-WAN and Enhanced SD-WAN during link failure scenarios using python generated packets are depicted in Figure 58.

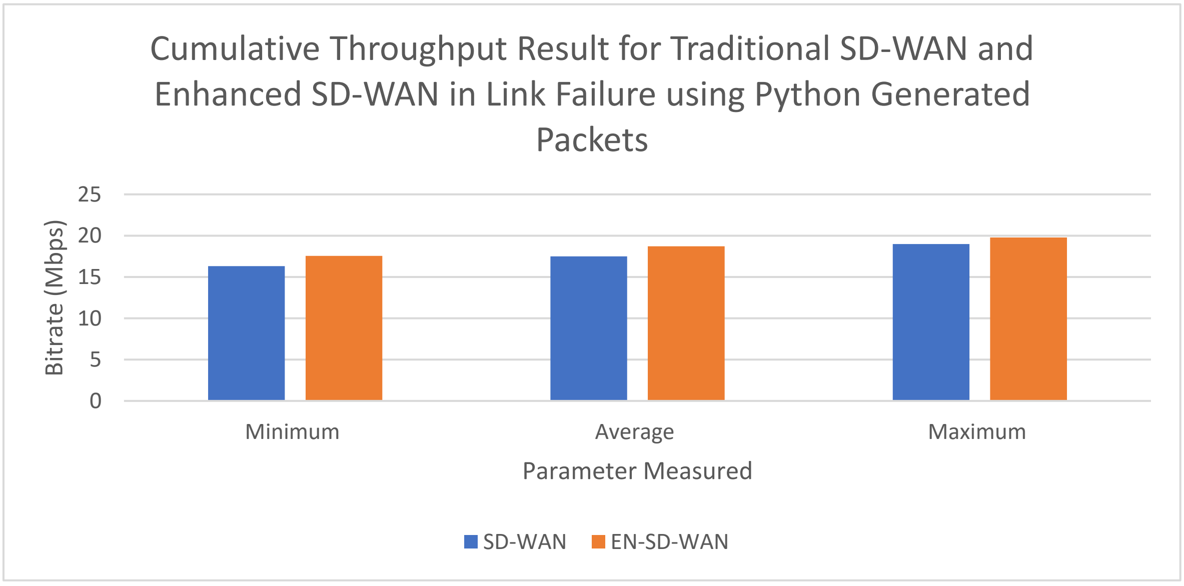


Figure 58: Cumulative Throughput result for Traditional SD-WAN and Enhanced SD-WAN in Link Failure

6.5.4: Link Failure Traffic Using GNS3 Application Traffic

This section examines the performance of the traditional SD-WAN (Non-Enhanced SD-WAN) and Enhanced SD-WAN architectures when a WAN link fails, using real-time traffic analysis. In this implementation the GNS3 applications are used to transfer the various data files from the various clients to the servers via the use of HTTP. Figure 59 displays the outcomes of the Enhanced and Traditional SD-WAN architecture revealing a notable improvement in throughput. The highest throughput recorded is above 24Mbps, while the lowest is noted as 21.98Mbps. These findings reflect elevated throughput levels, suggesting minimal protocol overhead and congestion within the network architecture.

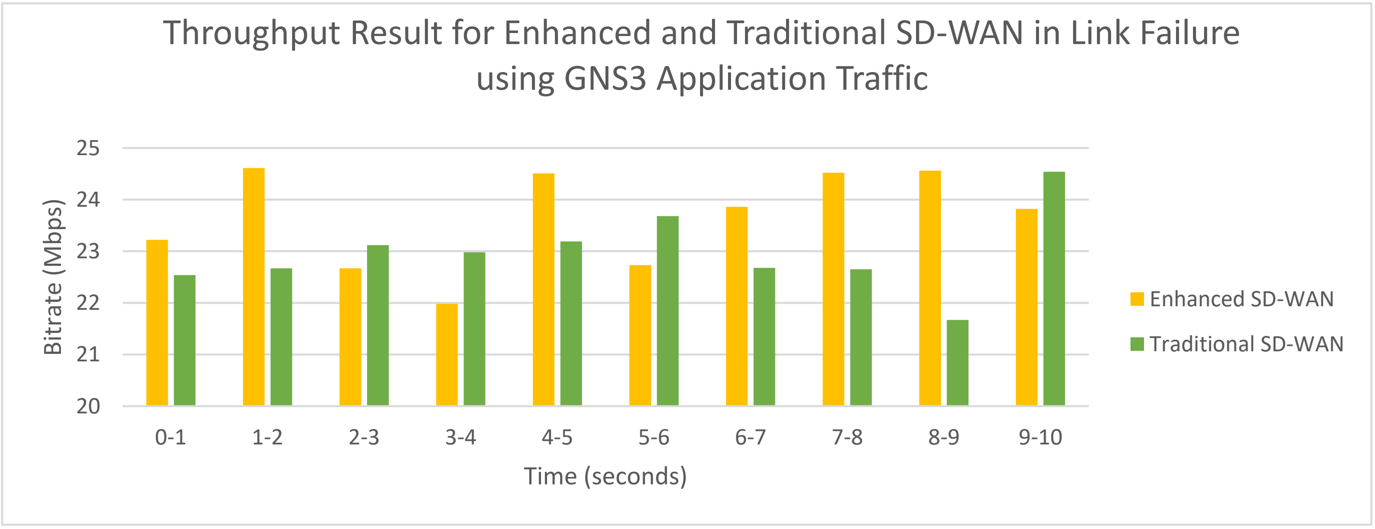


Figure 59: Throughput result for Enhanced and Traditional SD-WAN in Link Failure using GNS3 Application Traffic.

In contrast, in the traditional SD-WAN, when a link failure occurs, the throughput value decreases to below 24.54Mbps, representing the maximum throughput, with the lowest throughput recorded at 21.67Mbps. This decline in throughput performance is attributed to the deactivation of traffic management functionalities and the enhanced SD-WAN controller capabilities.

Furthermore, when comparing traditional SD-WAN and Enhanced SD-WAN performance during link failure, it becomes evident that Enhanced SD-WAN outperforms the traditional SD-WAN in terms of minimum, average, and maximum throughput. This observation indicates that Enhanced SD-WAN can swiftly converge upon the failure of any WAN link and seamlessly manage network traffic routing among the remaining WAN links. This underscores the importance of implementing effective traffic management policies that enable the controller to share routing pattern changes information for efficient load sharing using real-time traffic. Figure 60 illustrates the throughput results for traditional SD-WAN and Enhanced SD-WAN during link failure using real-time traffic.

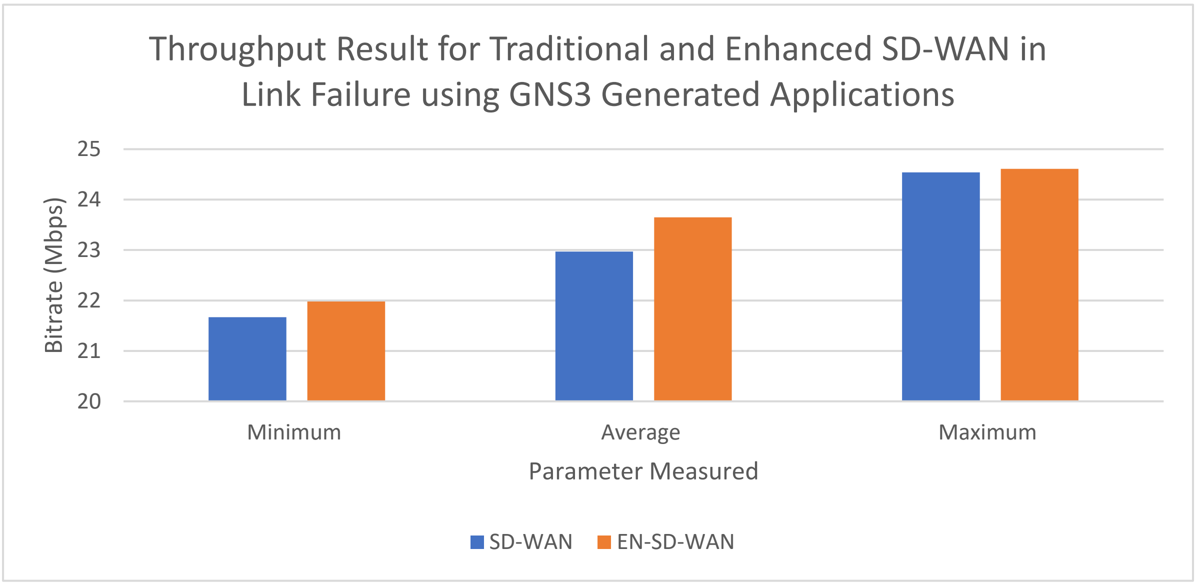


Figure 60: Cumulative Throughput result for Traditional SD-WAN and Enhanced SD-WAN in Link Failure using GNS3 generated Applications.

6.5.5: Congested Traffic Using Python Script

This section focused on introducing network congestion by reviewing the speed, bandwidth and MTU of the various routers connected in the network. Thus, the traffic load is maintained with same values while the bandwidth is reviewed to128 Kbps (kilobits per second) to create a bottleneck, the MTU to 1000 bytes, which is smaller than the default MTU of Ethernet networks (typically 1500 bytes) and 1 Mbps (megabits per second), which is lower than typical interface speeds. Based on these parameters, there is congestion experienced in the network WAN links and that has affected the throughput results obtained from the simulation. Figure 61 (results in orange) shows the throughput result for Enhanced SD-WAN when the network is congested.

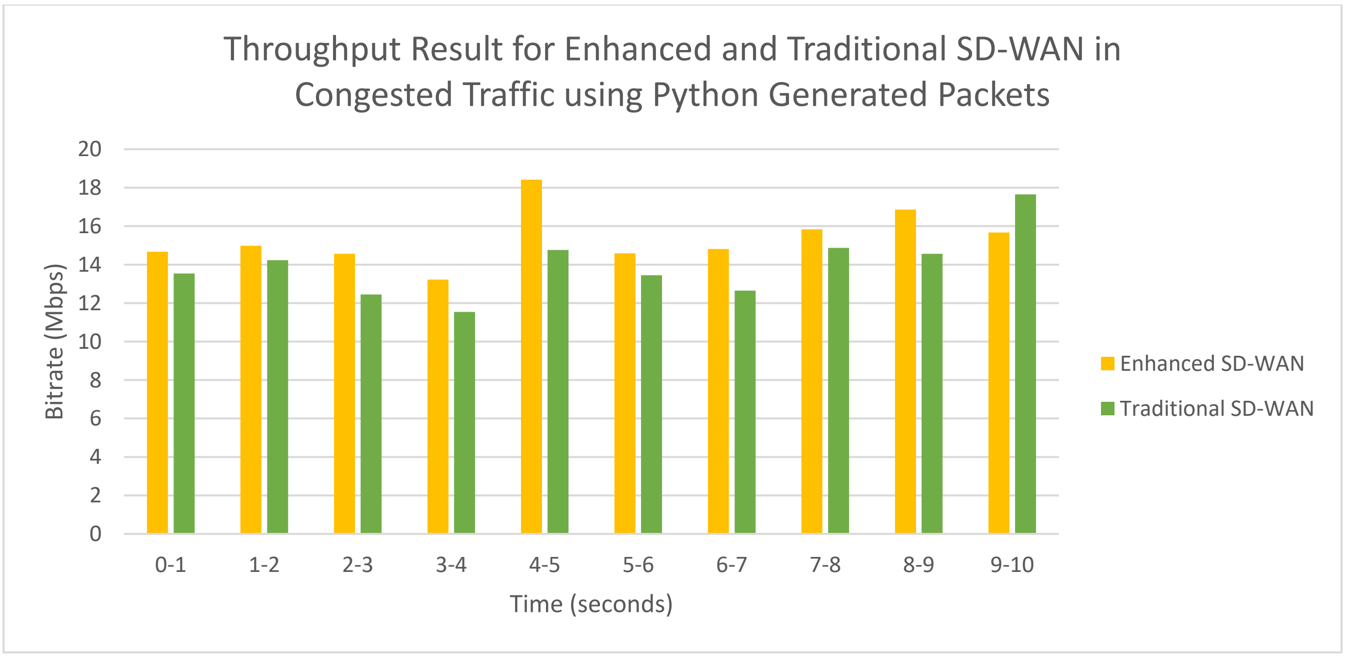


Figure 61: Throughput result for Enhanced and Traditional SD-WAN when network is congested.

In the results obtained in Figure 61, the throughput is less than 20Mbps in all the various time which is primary because of the congestion that is been introduced in the network. Also, it is important to mention the highest throughput is 18.42Mbps for the enhanced SD-WAN controller. Whilst Figure 61(results in green) shows the throughput result for Traditional SD-WAN when network is congested.

In contrary to the results obtained from the Enhanced SD-WAN, the traditional SD-WAN results shows lower performance in throughput. The maximum throughput is 17.65Mbps which is lower than 18.42Mbps compared to the Enhanced SD-WAN which is more than 18Mbps. A comparison of the Enhanced SD-WAN and Traditional SD-WAN in terms of the throughput performance is shown in Figure 62.

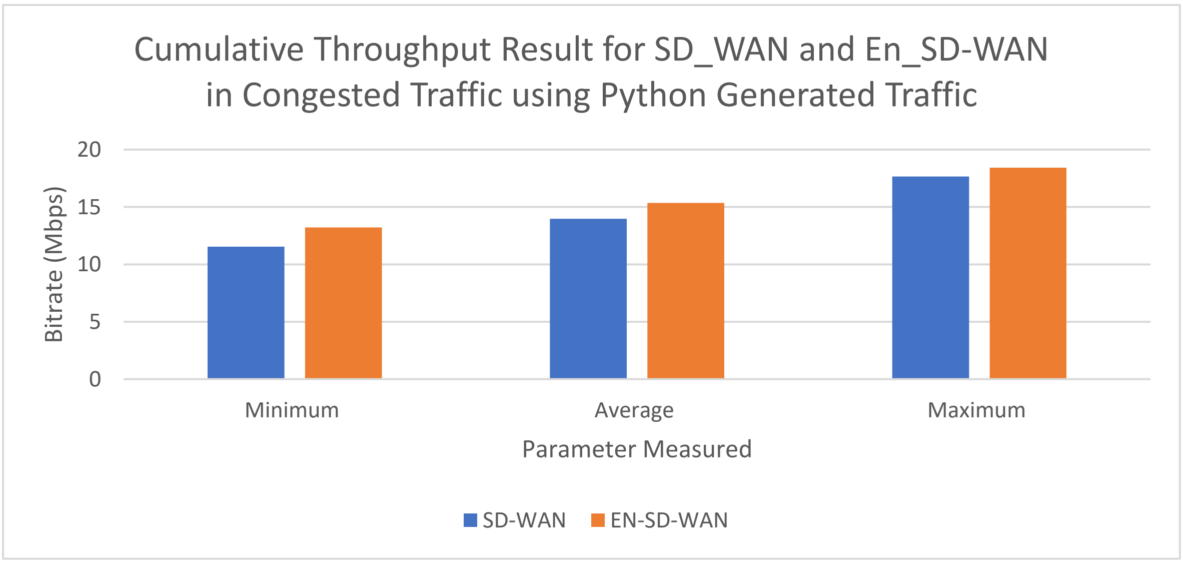


Figure 62: Cumulative Throughput result for SD-WAN and Enhanced SD-WAN in Congested Traffic.

Figure 62 illustrates consistent throughput results, demonstrating that Enhanced SD-WAN outperforms Traditional SD-WAN across all metrics - minimum, average, and maximum when utilising python generated packets. The throughput performance of Enhanced SD-WAN is notably higher at 52.37% compared to traditional SD-WAN's 47.63%. The difference between the Enhanced SD-WAN controller compared to the traditional SD-WAN controller is 4.74%. Hence, the slight difference in terms of throughput of 4.74% is because of the enhanced SD-WAN controller is better than the traditional SD-WAN because of the traffic management policies implemented about the enhanced SD-WAN controller. Moreover, it is crucial to acknowledge that network congestion can result in packet loss and a subsequent decrease in throughput of the general network performance which might be the case of the traditional SD-WAN controller.

6.5.6: Congested Traffic Using Packets Generated via GNS3 Applications

The use of congested traffic to assess the performance of SD-WAN and Enhanced SD-WAN architecture provides additional insights into the behaviour of the developed SD-WAN architecture. This section utilises same values of bandwidth is 128 Kbps (kilobits per second) to create a bottleneck, the MTU to 1000 bytes, which is smaller than the default MTU of Ethernet networks (typically 1500 bytes) and 1 Mbps (megabits per second), which is lower than typical interface speeds. Figure 63 depicts the throughput performance of Enhanced SD-WAN when network congestion is detected. The results indicate that the maximum throughput above 10Mbps and the lowest throughput was at the beginning of the transmission which is from 0 to 1 seconds using the packets generated via GNS3 Applications.

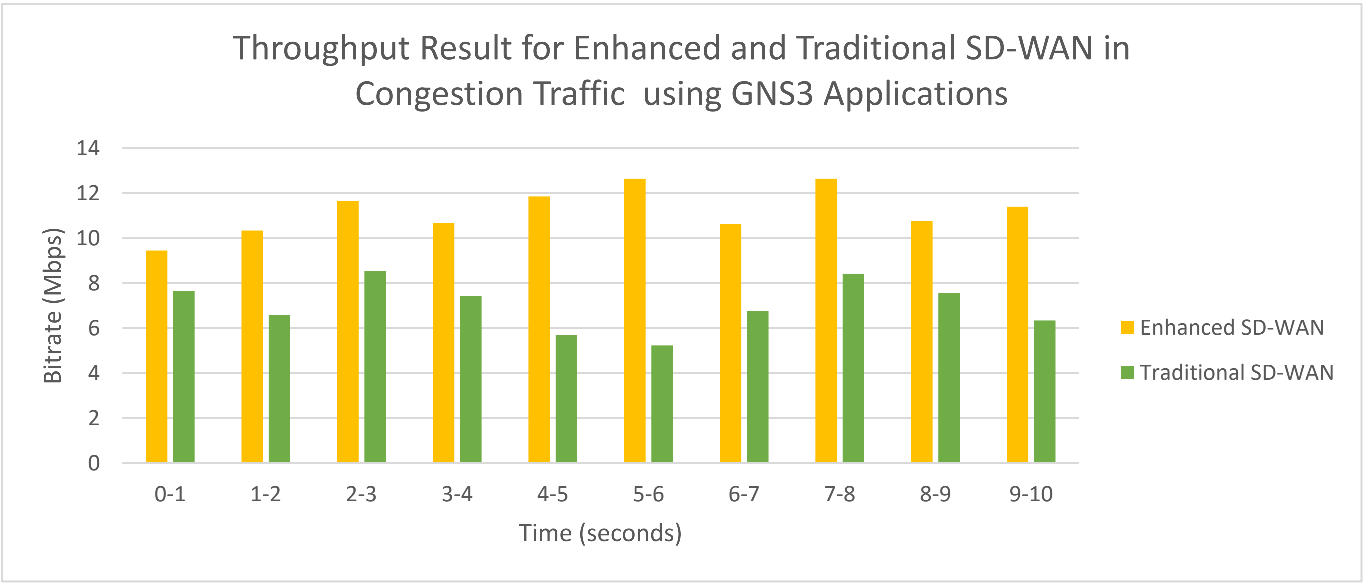


Figure 63: Throughput result for Enhanced and Traditional SD-WAN in Congested Traffic.

Furthermore, is when the same procedure is implemented when the traffic monitoring process is not included which is the traditional SD-WAN controller approach.

The findings from the traditional SD-WAN demonstrate a decline in throughput performance due to network congestion. In this architecture, the maximum throughput is consistently below 9Mbps, representing one of the lowest values observed across all throughput analyses. Figure 64 presents a comparison of throughput performance between Enhanced SD-WAN and SD-WAN, highlighting their respective performance in response to network congestion. The comparison indicates that Enhanced SD-WAN outperforms traditional SD-WAN in terms of throughput, showcasing its superior performance under congested network conditions.

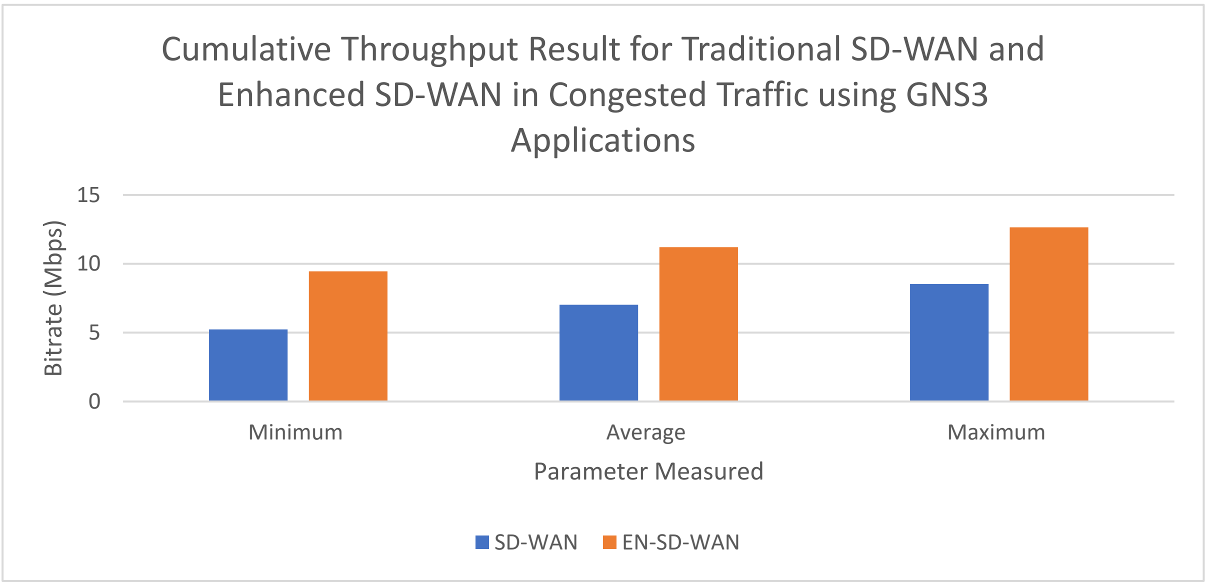


Figure 64: Cumulative Throughput result for Traditional SD-WAN and Enhanced SD-WAN in Congested Traffic.

6.6: Result and Design Validation

In this section, a comparative analysis is provided for the outcomes achieved through the implementation of the GNS3 application and the Python-based implementation. The GNS3-designed implementation serves as a validation tool for the Python implementation. The results demonstrate similarity between the outcomes derived from employing the GNS3 application and those from the Python-developed applications. Figure 65: describes the validation result of the Enhanced SD-WAN Architecture using a steady flow of traffic.

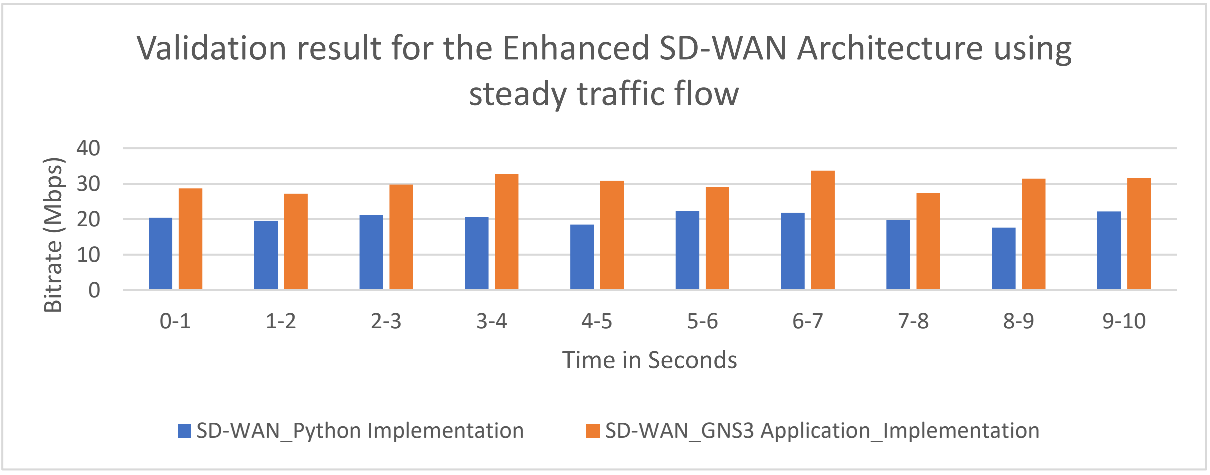


Figure 65: Describes the validation result of the Enhanced SD-WAN Architecture using a steady flow of traffic for throughput.

The results described in Figure 65 demonstrate a consistent throughput over the specified time duration, both in the GNS3 application and the Python-based implementation. This reaffirms the validity of the research design, which ensured consistent parameters across all routing and active devices during the design phase. However, it's noteworthy to acknowledge a slight deviation observed in the graph where the throughput result for GNS3 Application is at the highest point of about 32Mbps compared to the python of 22Mbp, attributed to technical bugs and update errors. Challenges such as route update, IOS image compatibility issues, and resource allocation constraints have been identified as potential factors contributing to these minor variations in the presented results.

Moreover, validation results are obtained upon implementation, particularly when encountering a network link failure. The findings of this evaluation are depicted in Figure 66. A comparison between the validation results for link failure using the GNS3 application and Python implementation reveals minor discrepancies attributed to slight variations in simulation duration. This disparity is evident in the graph displayed in Figure 66, where the maximum bitrate for the Enhanced SD-WAN using the GNS3 application is recorded as 24Mbps, whereas in the case of Python implementation, it stands at 20Mbps. The marginal variance in throughput ratings could possibly stem from technical glitches within the simulator.

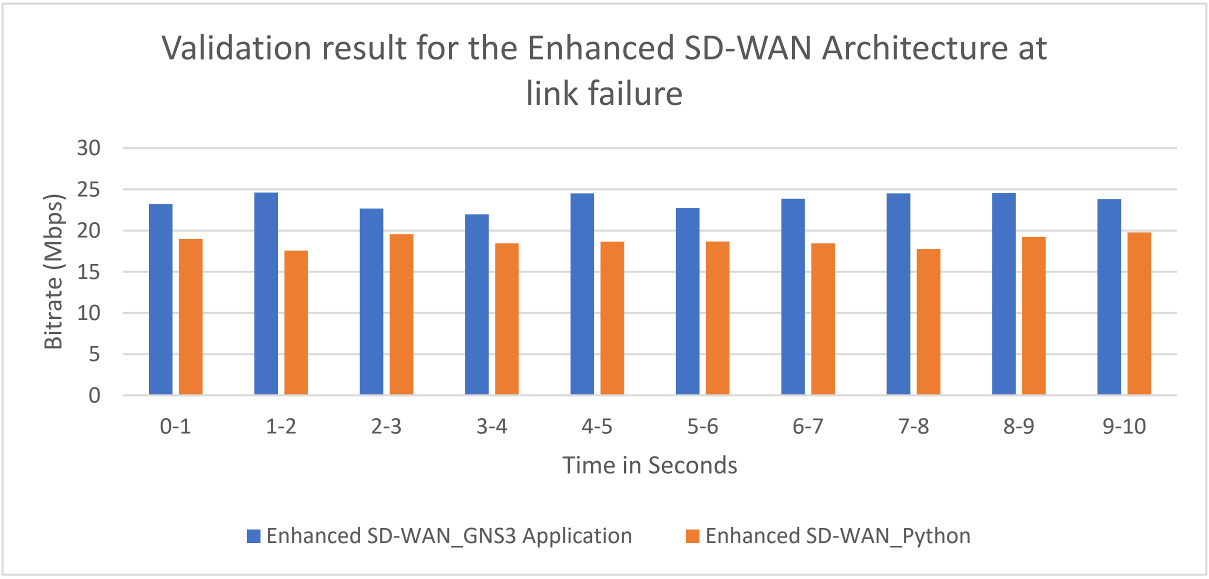


Figure 66: Describes the validation result of the Enhanced SD-WAN Architecture at the failure of a link for GNS3 Application and Python implementation.

Also, it is the validation performance when the network is congested in the network architecture when the GNS3 application and python is implemented. Based on the result the GNS3 application have slight difference in performance compared to the python application. Possible reasons for these trends include escalating network congestion as bit rate rises, resulting in overall network performance slowdowns affecting the validation processes, as well as increased resource utilization in the validation process with higher bitrates, potentially leading to bottlenecks. Figure 67 shows the results of the Enhanced SD-WAN architecture implementation using GNS3 and python applications when the network is congested.

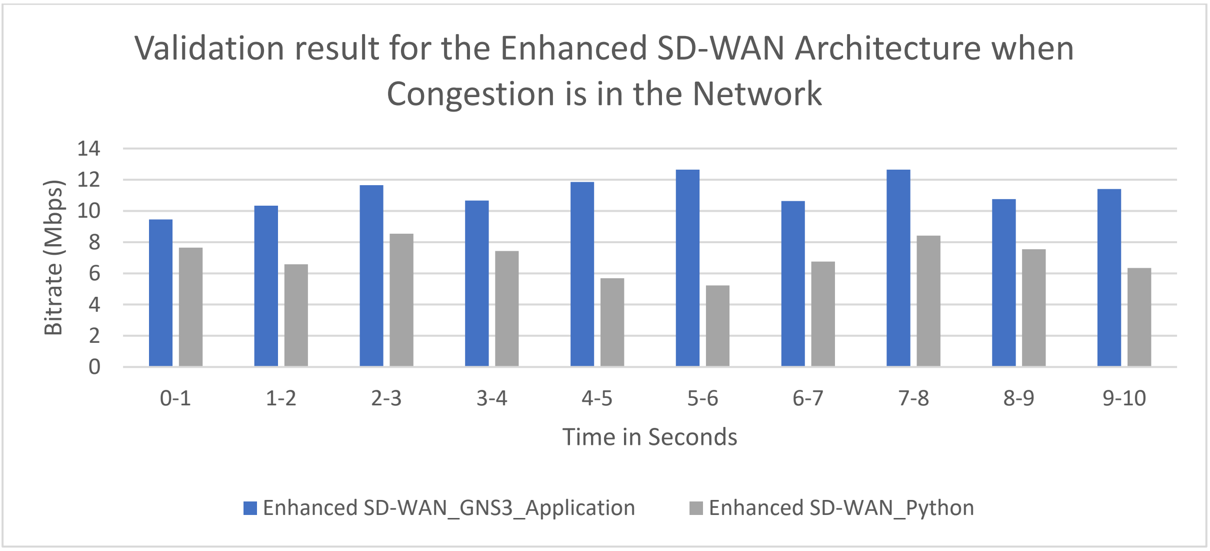


Figure 67: Validation result for the Enhanced SD-WAN Architecture when Congestion is in the Network

6.7: Analysis of Result Presentation

In the review of the first scenario presented which is normal traffic transmission, both Enhanced SD-WAN and traditional SD-WAN were tested for throughput. Enhanced SD-WAN exhibited similar throughput rates, ranging from 20Mbps to 22Mbps, indicating efficient data transmission. Moreover, in terms of bitrate over time, both traditional SD-WAN and Enhanced SD-WAN exhibited similar performance, with a throughput ranging from 20Mbps to 22Mbps. A comparison between the two models indicated that Enhanced SD-WAN slightly outperformed traditional SD-WAN, achieving a 50.07% higher overall bitrate compared to the latter's 49.93%. This difference highlights Enhanced SD-WAN's marginally superior performance. It is evident that under normal conditions, without congestion, both controllers effectively handle the transmission of data.

The evaluation extended to the use of GNS3 applications traffic for data transfer whereby Enhanced SD-WAN demonstrated an average throughput consistently, albeit with fluctuations over time, while traditional SD-WAN maintained consistent with similar throughput across the various time. It is important to mention in both Python and GNS3 generated packets across the network both scenarios have consistent performance in terms of throughput, and this is because there is no congestion experienced in the network. Enhanced SD-WAN is consistent with the traditional SD-WAN in throughput metrics, across performance in maximum, minimum, and average throughput.

In the assessment of link failure scenarios using python generated packets, both Enhanced SD-WAN and traditional SD-WAN were evaluated to understand how the network handles traffic redistribution among WAN links. Enhanced SD-WAN showcased higher throughput, with maximum rates of 19.78Mbps and minimum rates of 17.56Mbps during link failure. Equally, traditional SD-WAN exhibited a decrease in throughput, with maximum rates of 18.98Mbps and minimum rates of 16.32Mbps, attributed to the lack of functionalities in its traffic management policy. The analysis highlighted Enhanced SD-WANs has slightly better performance in terms of minimum, average, and maximum throughput compared to traditional SD-WAN, emphasising the importance of robust traffic management policies for efficient network convergence following WAN link failures.

Furthermore, when analysing link failure scenarios using GNS3 applications traffic, Enhanced SD-WAN demonstrated improved throughput compared to traditional SD-WAN. Enhanced SD-WAN recorded throughput values ranging to 24.61Mbps, with minimal protocol overhead and congestion, while traditional SD-WAN exhibited decreased throughput value of 24.53Mbps, indicating the slight impact of deactivating the traffic management functionalities. The comparison between traditional SD-WAN and Enhanced SD-WAN underscored Enhanced SD-WAN's ability to swiftly converge and manage network traffic routing among remaining WAN links, highlighting the necessity of effective traffic management policies for efficient load sharing using real-time traffic.

Also, the assessment of network congestion using python generated packet focused on evaluating how the Enhanced SD-WAN architecture dynamically adjusts traffic routing to reduce congestion while maintaining performance. Enhanced SD-WAN demonstrated throughput values consistently above 15Mbps during congestion periods, indicating improved performance in managing network traffic load. In contrast, traditional SD-WAN exhibited lower throughput, with maximum rates around 13Mbps, highlighting the superior performance of Enhanced SD-WAN in handling congested traffic. Figure 67 illustrates a comparison between SD-WAN and Enhanced SD-WAN in terms of throughput, with Enhanced SD-WAN outperforming SD-WAN across all metrics, emphasising the effectiveness of its traffic management policies and controller functionalities in optimizing network performance.

Furthermore, the analysis of congested traffic using GNS3 applications packets provided additional insights into the behaviour of SD-WAN architectures. Enhanced SD-WAN demonstrated a drop in throughput during network congestion, with maximum throughput falling below 15Mbps. On the contrary, traditional SD-WAN exhibited consistently lower throughput values below 10Mbps, reflecting its inferior performance under congested conditions. The comparison between Enhanced SD-WAN and traditional SD-WAN highlighted Enhanced SD-WAN's superior throughput performance, emphasising its effectiveness in managing network congestion and maintaining higher throughput levels compared to traditional SD-WAN architecture.

6.8: Conclusion

Chapter six focuses on the results obtained. These confirm the various objectives proposed in chapter one which involves having a new logical architecture that could modify the route when there is an indication of congestion in any of the WAN links.

CHAPTER SEVEN: CONCLUSION AND RECOMMENDATIONS

Achieving the fundamental goals of this SD-WAN and SDN research will enhance and expand the existing knowledge in the field. The objectives of this research have been methodically addressed from chapter one to the sixth chapter of this research. Each chapter has played a crucial role in both identifying the knowledge gaps and making valuable contributions to the current understanding of SD-WAN and SDN.

7.1: Overview

This research starts with the concept and introduction of SD-WAN and SDN with the primary objective of creating new logical architecture for the SD-WAN architecture to facilitate dynamic load balancing. This is achieved by deep packet inspection of end-to-end protocol performance which facilitate the dynamic sharing of load in real time. This final chapter is centred on summarising the research and providing recommendations. A comprehensive summary of the entire research, highlighting the knowledge contributions derived from the preceding research and literature, particularly discussed in chapter three of this study.

7.2: Conclusion

The research question for this thesis is focused on the ability of SD-WAN load balancing's capability to be improved by dynamically analysing the end-to-end protocol performance. In providing an answer to this research question, the relationship between the traffic management system decisions in SD-WAN and the end-to-end performance operation is examined. Based on an extensive review of available literatures, a new logical architecture for SD-WAN (modified SD-WAN architecture) in which a traffic monitoring process (classification, management, and analysis) is introduced between the SD-WAN edge routers and controllers. The traffic monitor provides a deep packet inspection on end-to-end application traffic and provides the SD-WAN controller with status updates which forms a new input to its load balancing algorithm in real time. These updates allow the SD-WAN to respond dynamically to the demands of applications and deliver an improved quality of service for the applications being transmitted.

The first objective of this research is the modification of the existing SD-WAN architecture to introduce a Traffic monitor function between the edge routers and SD-WAN controller for enabling dynamic response to end-to-end applications based on real-time requirements. This research objective has been achieved based on the network design implemented in section 5.2, 5.3 and 5.4 which represents the functional, operational, and schematic design of the new logical architecture SD-WAN implemented in this research. Therefore, the network design for this research implemented in section 5.5 is an affirmation of the implementation of a logical architecture which differs from the existing or current SD-WAN architecture. It is also important to recognise that the current design of SD-WAN architecture does not have a monitoring system function built distinctively from the controller, rather have a management plane depending on the vendor which has monitoring functions to show the health and visibility of the network performance metrics as suggested by(Juniper-Networks, 2023a; Kharub, 2022; PaloAlto-Networks, 2023).

The first objective is further developed to formulate the second objective of this research. The process involves conducting a Deep Packet Inspection (DPI) of the end-to-end traffic flows to gain insight into the network performance and perform the best routing approach by the SD-WAN controller. The DPI approach is used because of the provisions of advanced traffic analysis, prioritisation of managed flows and optimising network performance based on some specific network applications such as TCP. Thus, in this analysis using the DPI, the monitor has a knowledge of the window size, packet lengths, source/destination IP address, port numbers and congestion flags used in this research. Moreover, a closely related research which involves the use of monitoring tools at end-to-end as proposed by this research is demonstrated by (Navarro et al., 2023). In the research of (Navarro et al., 2023) and recent integration of the Telco Network Cloud Manager to SD-WAN architecture, identifies the need of having a monitoring device in the end-to-end protocol performance. Also, the current architecture of SD-WAN monitoring approach does not use private connections between sites, rather rely on internet connections from telecommunication providers which most times are reluctant to share real time traffic information of routing rules, link status and many more. This makes it difficult to have a complete picture of the network performance metrics and troubleshoot accordingly. Thus, this objective aims to use a network monitoring approach that has little or no interference with external factors and estimate performance metrics such as source/destination IP addresses, port number and congestion flags. This information is shared with the En-SDWAN controller which has a knowledge of the bandwidth assigned to various WAN links, the bandwidth capacity utilised by each WAN link and capacity of bandwidth left for new traffic flow to utilise in the result of congestion. This defined the requirement of the new logical architecture SD-WAN controller to have the required capability to re-route traffic flows in the possibility of network congestion.

The third objective is achieved by effectively and dynamically adapting to load balancing based on the summarised information obtained from the traffic management procedure of the monitor. Based on the findings outlined in sections 6.4, 6.5, and 6.6, the behaviour of both traditional and Enhanced SD-WAN architectures has been thoroughly examined. Section 6.4 focused on presenting results in a functional manner, detailing the entire process of packet monitoring and classification. These results underscored the Enhanced SD-WAN's capacity and functionality in terms of efficient route updates, equitable distribution of network traffic across multiple WANs, and effective bandwidth utilisation. In section 6.5, the performance of the developed Enhanced SD-WAN architecture was assessed in reference to the traditional SD-WAN across various scenarios, including normal traffic, link failure, and network congestion. Under normal traffic conditions, the Enhanced SD-WAN is like the traditional SD-WAN as the difference between performance is less than 1%. Moreover, during network link failures, the Enhanced SD-WAN exhibited quicker convergence and slight throughput performance compared to the traditional SD-WAN by 1.46% in GNS3 applications. Similarly, in instances of network congestion, the Enhanced SD-WAN demonstrated an average throughput of 11.21Mbps, surpassing the SD-WAN architecture's average throughput of 7.02Mbps. These results collectively suggest that adopting the new logical architecture with an Enhanced SD-WAN controller can enhance functionality within an SD-WAN environment, thereby improving quality of service and facilitating more efficient load sharing among various WAN networks, ultimately leading to enhanced throughput.

In essence, the novelty in this Thesis involves the design of a new logical architecture for SD-WAN with a traffic monitor connected between the edge routers which sends traffic updates in terms of congestion flags to En-SDWAN controller to facilitate dynamic load sharing. The traffic monitors are connected to the two edge routers to ensure end-to-end performance is achieved and the traffic updates are sent to the En-SDWAN controller and route updates are forwarded to respective edge routers in real-time. The implementation and optimisation of this mechanism is believed to result in better throughput and quality of service for the various users of SD-WAN.

7.3: Recommendations

Since the inception of SDN, network automation has attracted the attention of many researchers in the networking and telecommunication field leading to the invention of technology such as SD-WAN, NFV, Network Slicing and many more. This evolution in network development and application has various benefits and challenges that need to be addressed with the aim of improving quality of service and quality of experience by users. In this research specific focus is driven to SD-WAN in context to the aim of facilitating dynamic load balancing amongst WAN networks based on a new logical architecture. Thus, few of the areas that need improvement in this new logical architecture of SD-WAN are described below.

The monitoring process is evolving as network requirements and the number of devices connected to the internet are increasing daily. In this research, a DPI approach is applied in the traffic monitoring process connected to the En-SDWAN controller. However, this can be improved by ensuring a robust, comprehensive, and policy-based approach with a fine-grained holistic view of the entire traffic flow in the network. To achieve this, there is a necessity to apply and use the principle of Machine Learning Algorithms integrated with the application of Artificial Intelligence. It is expected the monitoring policies should be comprehensive, real-time, learn to adapt quickly to changes with network requirements and are scalable to accommodate variants of network requirements. Furthermore, it is essential to consider the operation of applications and protocols that operate above the TCP layer. Also, it is the issue of scalability and the ability of the monitor to adapt with a high level of growing network parameters and the birth of new corporate sites that need to be connected to the SD-WAN architecture. Scalability is vital in the part of the SD-WAN controller based on the increase in the number of flows, switches, and bandwidth requirement. In this research design implemented, it can be argued that all network flows captured by the traffic monitor are not forwarded to the En-SDWAN controller. But imagine a condition consisting of more than 4 million virtual machines that can generate 30 million flows of packet per second that leads to high levels of congestion and scalability issues. It is nearly impossible for a single controller to manage this volume of packets or flows which will lead to controller overhead. Thus, it is vital to consider further research in the scalability and volume of packets or flows that need to be transmitted from the monitor to the controller with the aim of avoiding controller overhead and issues of scalability of both the traffic monitor and En-SDWAN controller.

Network security is very important to ensure confidentiality, availability, and integrity of network flows amongst various WANs in the SD-WAN architecture. This research has not considered the technical security challenges (access control) and cost in the implementation of this SD-WAN technology. Security is a major concern in every evolving network architecture, with the increase in scalability of the network design and IoT devices combined with many primary sites that need to be connected will certainly be a concern. This research has not focused into the issues that concerns security around the monitor and En-SDWAN controller as this is a major concern based on the connected volume of WAN sites. For example, if the controller is deployed in a real-world scenario where access control is not limited, it can affect the algorithms design and can affect the routing updates sent to the edge routers. Thus, it is recommended to have further research into the security architecture of both the monitor and En-SDWAN controller proposed in the new logical architecture to meet all the standards to avoid any form of security downtime.

One of the persistent issues that comes with the implementation of SD-WAN architecture is the interoperability with legacy networks which was even observed at the implementation of this research architecture using GNS3. It is vital that the SD-WAN architecture needs to integrate seamlessly with the existing traditional networks at the various edge sites and telcos (telco-operators) on end-to-end. But this has not been the case with the SD-WAN architecture and achieving the process of integration is a gradual process and might take time for the entire legacy system to integrate with the SD-WAN system. Because most of the enterprise network might want to consider the cost of integrating SD-WAN features, training of team members and upgrading most of the equipment to adapt to these changes required in the implementation of SD-WAN.

7.4: Further Works

The developed solution still requires further work based on the nature of the data being transmitted between the client and the server. For example, financial data or applications should not be routed over the internet leading to the need for a more sophisticated algorithm. Under these conditions there is a need to move some non-congested flows away from the WAN link being used by the financial application to free up some more space for it and hence, eliminate the congestion it is experiencing. The reason for this specific consideration is based on security concerns over the internet for heavy or risk financial applications which might be exposed or possess to be a major risk for the entire architecture.

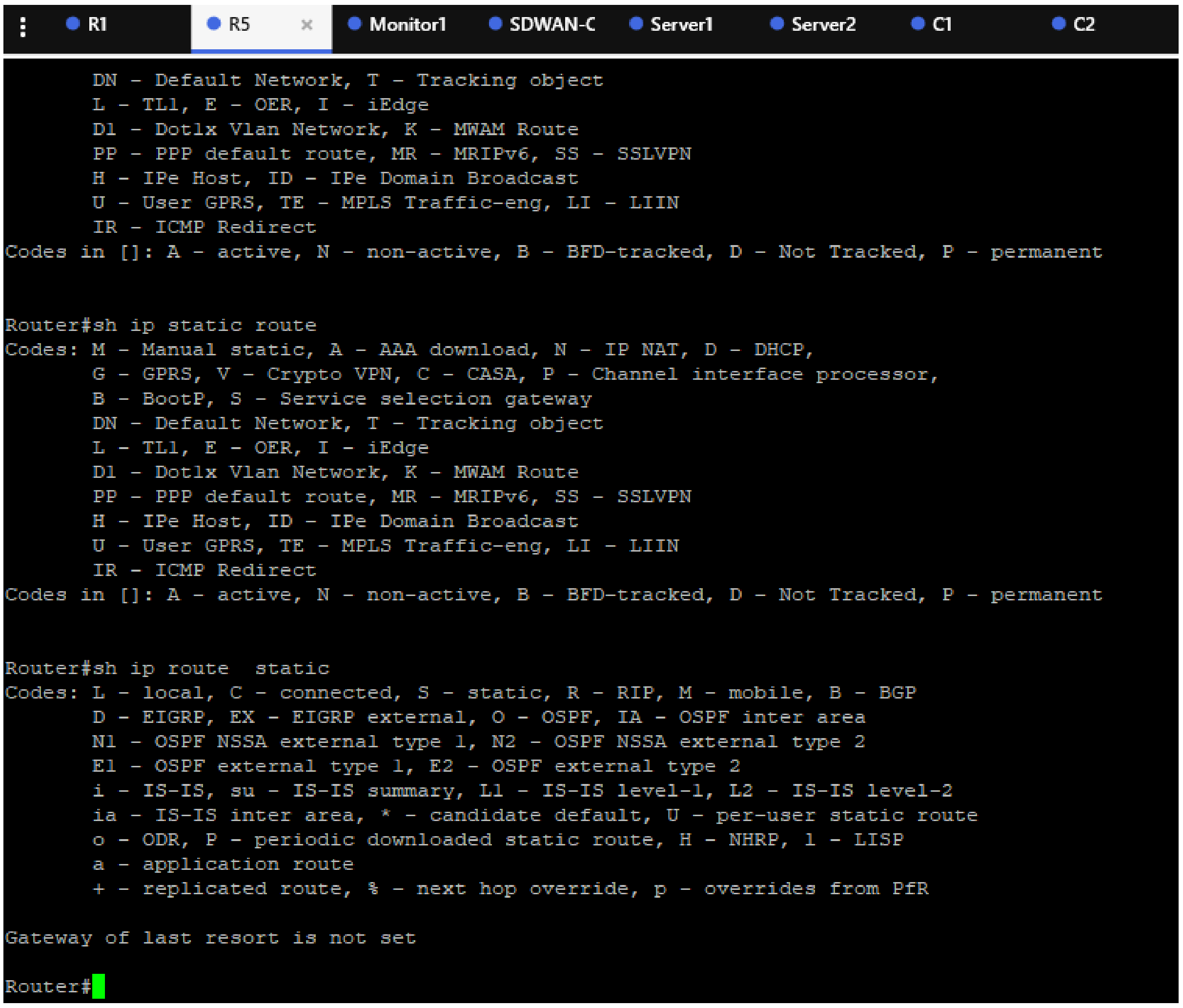
Also, it is the need to define hysteresis in the congested flows to avoid issues of excessive route flipping. In the developed algorithm, the route change is more frequent which might lead to route flipping. Hence, there is a need to have further work in the developed algorithm to create hysteresis. This will ensure that routes for changing congestion can only be affected within a period which will lead to stability of the network flows. This is very vital because of the stability of the entire SD-WAN architecture. This is because network instability can lead to variants of issues such as downtime, poor user experience and data loss.

Furthermore, it is a compliance and regulatory process in deploying this new SD-WAN architecture in real time network architecture. This is based on the concerns of telecommunication companies' restrictions and the need for service level agreement incorporating the traffic monitor function with service providers to effectively monitor the entire network flows. These levels of agreement are often required for an effective deployment of such architecture. Also, as established in the literature there is a need for compliance and standardisation with the MEF (Metro Ethernet Forum) to ensure such new architecture is fit for purpose and is modelled to improve customer user experience.

Appendix 1: Edge routers Result Presentation

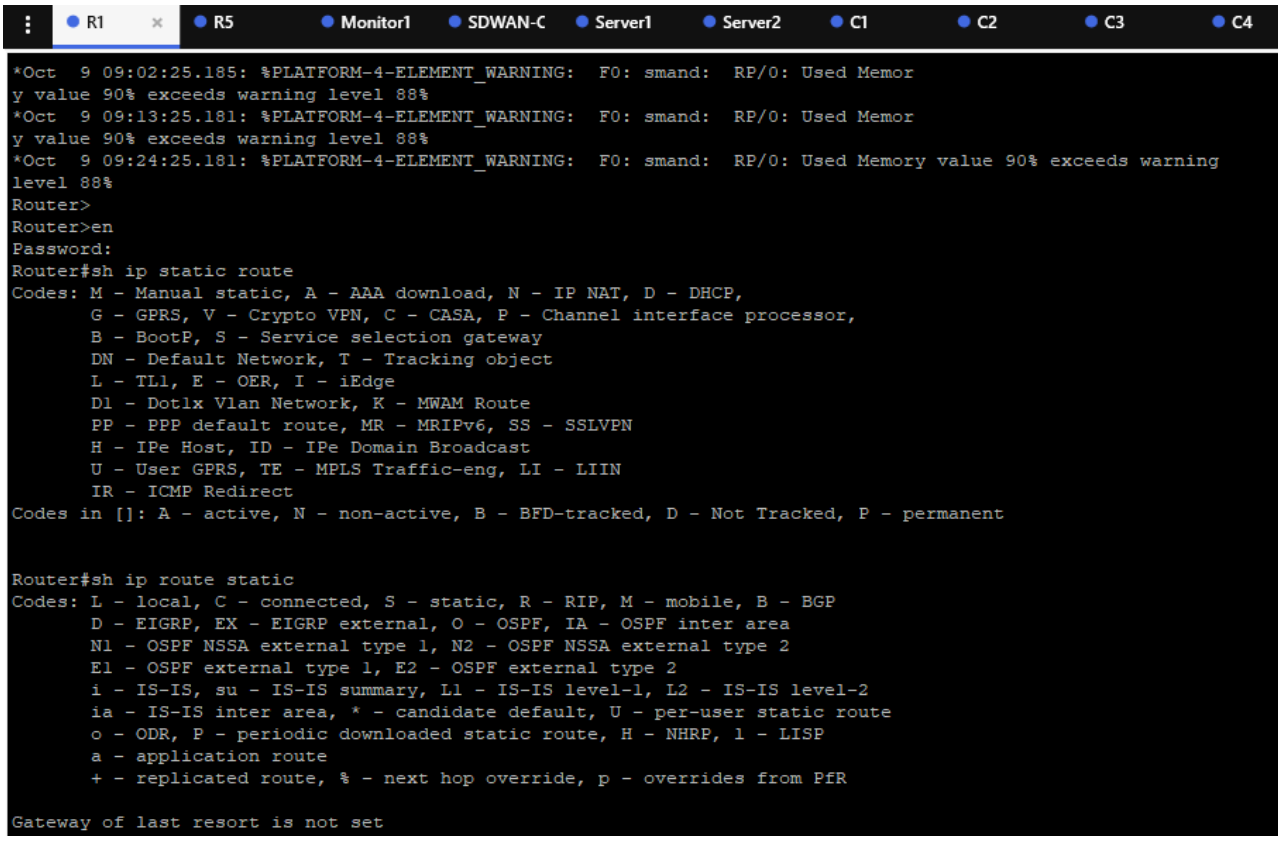
The initial configuration showing no route configuration for both edge router one and two.

➢ Edge router one showing no route configuration.



```
       DN - Default Network, T - Tracking object
       L - TL1, E - OER, I - iEdge
       D1 - Dot1x Vlan Network, K - MWAM Route
       PP - PPP default route, MR - MRIPv6, SS - SSLVPN
       H - IPe Host, ID - IPe Domain Broadcast
       U - User GPRS, TE - MPLS Traffic-eng, LI - LIIN
       IR - ICMP Redirect
Codes in []: A - active, N - non-active, B - BFD-tracked, D - Not Tracked, P - permanent


Router#sh ip static route
Codes: M - Manual static, A - AAA download, N - IP NAT, D - DHCP,
       G - GPRS, V - Crypto VPN, C - CASA, P - Channel interface processor,
       B - BootP, S - Service selection gateway
       DN - Default Network, T - Tracking object
       L - TL1, E - OER, I - iEdge
       D1 - Dot1x Vlan Network, K - MWAM Route
       PP - PPP default route, MR - MRIPv6, SS - SSLVPN
       H - IPe Host, ID - IPe Domain Broadcast
       U - User GPRS, TE - MPLS Traffic-eng, LI - LIIN
       IR - ICMP Redirect
Codes in []: A - active, N - non-active, B - BFD-tracked, D - Not Tracked, P - permanent


Router#sh ip route  static
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is not set

Router#
```

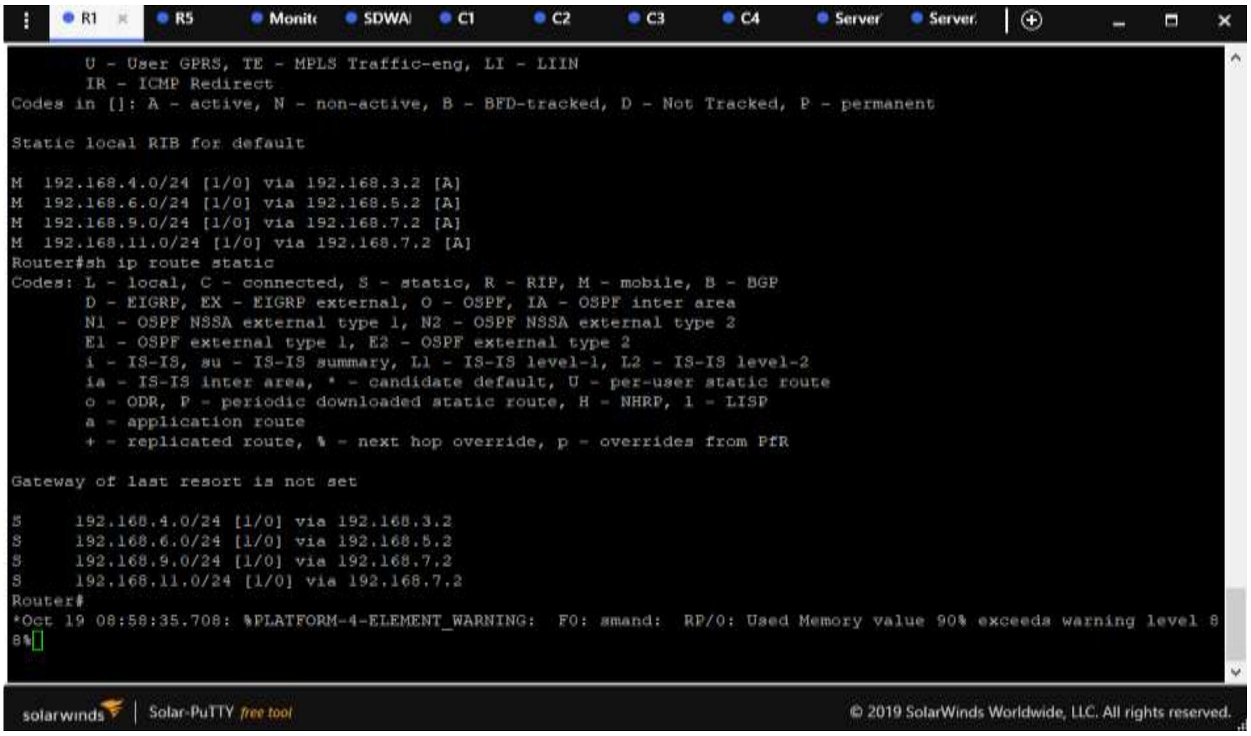
➢ Edge router two showing no route configuration.



```
*Oct  9 09:02:25.185: %PLATFORM-4-ELEMENT_WARNING:  F0: smand:  RP/0: Used Memor
y value 90% exceeds warning level 88%
*Oct  9 09:13:25.181: %PLATFORM-4-ELEMENT_WARNING:  F0: smand:  RP/0: Used Memor
y value 90% exceeds warning level 88%
*Oct  9 09:24:25.181: %PLATFORM-4-ELEMENT_WARNING:  F0: smand:  RP/0: Used Memory value 90% exceeds warning
level 88%
Router>
Router>en
Password:
Router#sh ip static route
Codes: M - Manual static, A - AAA download, N - IP NAT, D - DHCP,
       G - GPRS, V - Crypto VPN, C - CASA, P - Channel interface processor,
       B - BootP, S - Service selection gateway
       DN - Default Network, T - Tracking object
       L - TL1, E - OER, I - iEdge
       D1 - Dot1x Vlan Network, K - MWAM Route
       PP - PPP default route, MR - MRIPv6, SS - SSLVPN
       H - IPe Host, ID - IPe Domain Broadcast
       U - User GPRS, TE - MPLS Traffic-eng, LI - LIIN
       IR - ICMP Redirect
Codes in []: A - active, N - non-active, B - BFD-tracked, D - Not Tracked, P - permanent


Router#sh ip route static
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is not set
```

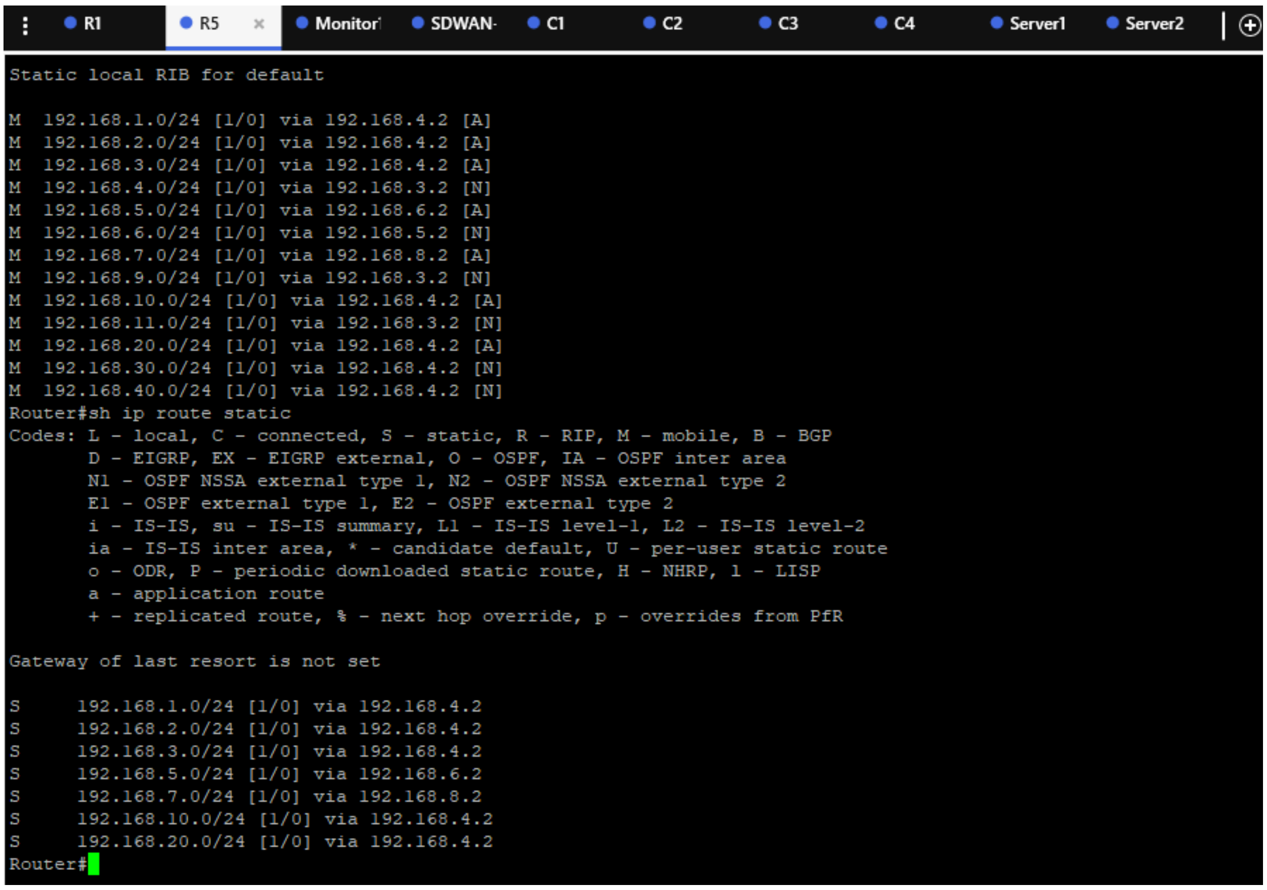
Appendix 2: Edge routers Result Presentation

➢ Edge router one showing route configuration.



```
       U - User GPRS, TE - MPLS Traffic-eng, LI - LIIN
       IR - ICMP Redirect
Codes in []: A - active, N - non-active, B - BFD-tracked, D - Not Tracked, P - permanent

Static local RIB for default

M  192.168.4.0/24 [1/0] via 192.168.3.2 [A]
M  192.168.6.0/24 [1/0] via 192.168.5.2 [A]
M  192.168.9.0/24 [1/0] via 192.168.7.2 [A]
M  192.168.11.0/24 [1/0] via 192.168.7.2 [A]
Router#sh ip route static
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is not set

S     192.168.4.0/24 [1/0] via 192.168.3.2
S     192.168.6.0/24 [1/0] via 192.168.5.2
S     192.168.9.0/24 [1/0] via 192.168.7.2
S     192.168.11.0/24 [1/0] via 192.168.7.2
Router#
*Oct 19 08:58:35.708: %PLATFORM-4-ELEMENT_WARNING:  F0: smand:  RP/0: Used Memory value 90% exceeds warning level 8
8%
```

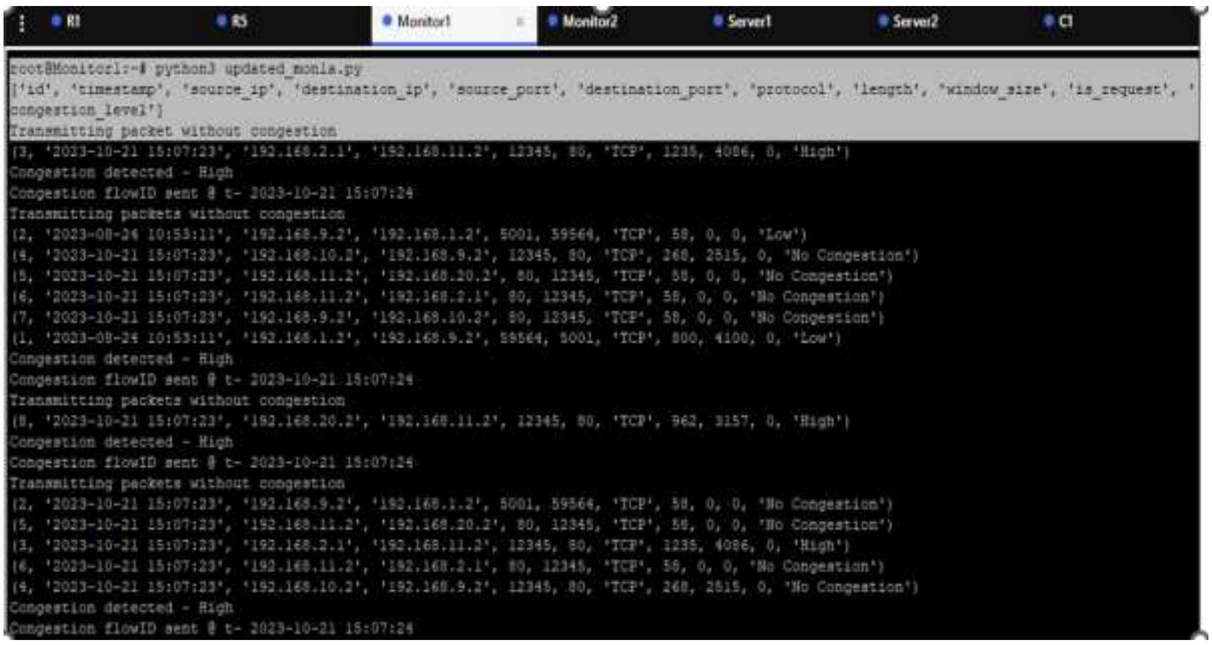
➢ Edge router two showing route configuration.



```
Static local RIB for default

M  192.168.1.0/24 [1/0] via 192.168.4.2 [A]
M  192.168.2.0/24 [1/0] via 192.168.4.2 [A]
M  192.168.3.0/24 [1/0] via 192.168.4.2 [A]
M  192.168.4.0/24 [1/0] via 192.168.3.2 [N]
M  192.168.5.0/24 [1/0] via 192.168.6.2 [A]
M  192.168.6.0/24 [1/0] via 192.168.5.2 [N]
M  192.168.7.0/24 [1/0] via 192.168.8.2 [A]
M  192.168.9.0/24 [1/0] via 192.168.3.2 [N]
M  192.168.10.0/24 [1/0] via 192.168.4.2 [A]
M  192.168.11.0/24 [1/0] via 192.168.3.2 [N]
M  192.168.20.0/24 [1/0] via 192.168.4.2 [A]
M  192.168.30.0/24 [1/0] via 192.168.4.2 [N]
M  192.168.40.0/24 [1/0] via 192.168.4.2 [N]
Router#sh ip route static
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is not set

S     192.168.1.0/24 [1/0] via 192.168.4.2
S     192.168.2.0/24 [1/0] via 192.168.4.2
S     192.168.3.0/24 [1/0] via 192.168.4.2
S     192.168.5.0/24 [1/0] via 192.168.6.2
S     192.168.7.0/24 [1/0] via 192.168.8.2
S     192.168.10.0/24 [1/0] via 192.168.4.2
S     192.168.20.0/24 [1/0] via 192.168.4.2
Router#
```

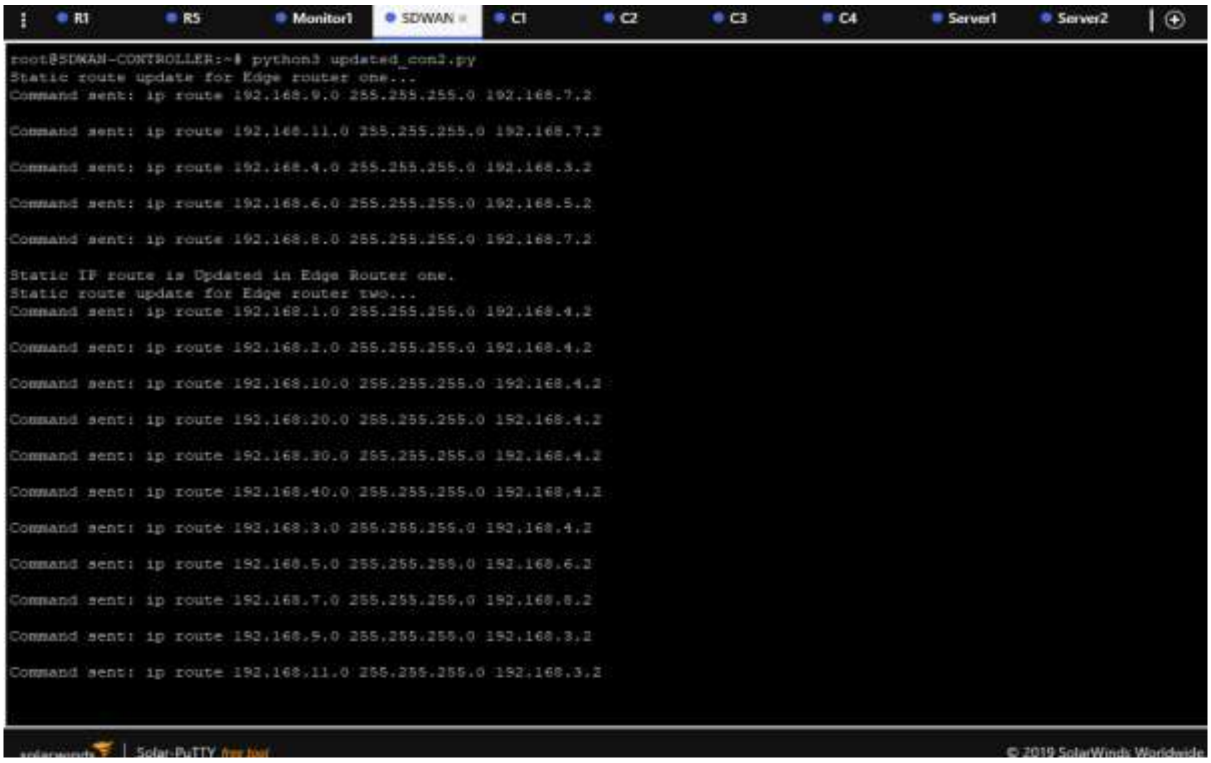
Appendix 3: Monitor and SD-WAN Controller Presentation

➢ The initilisation of the Monitor system



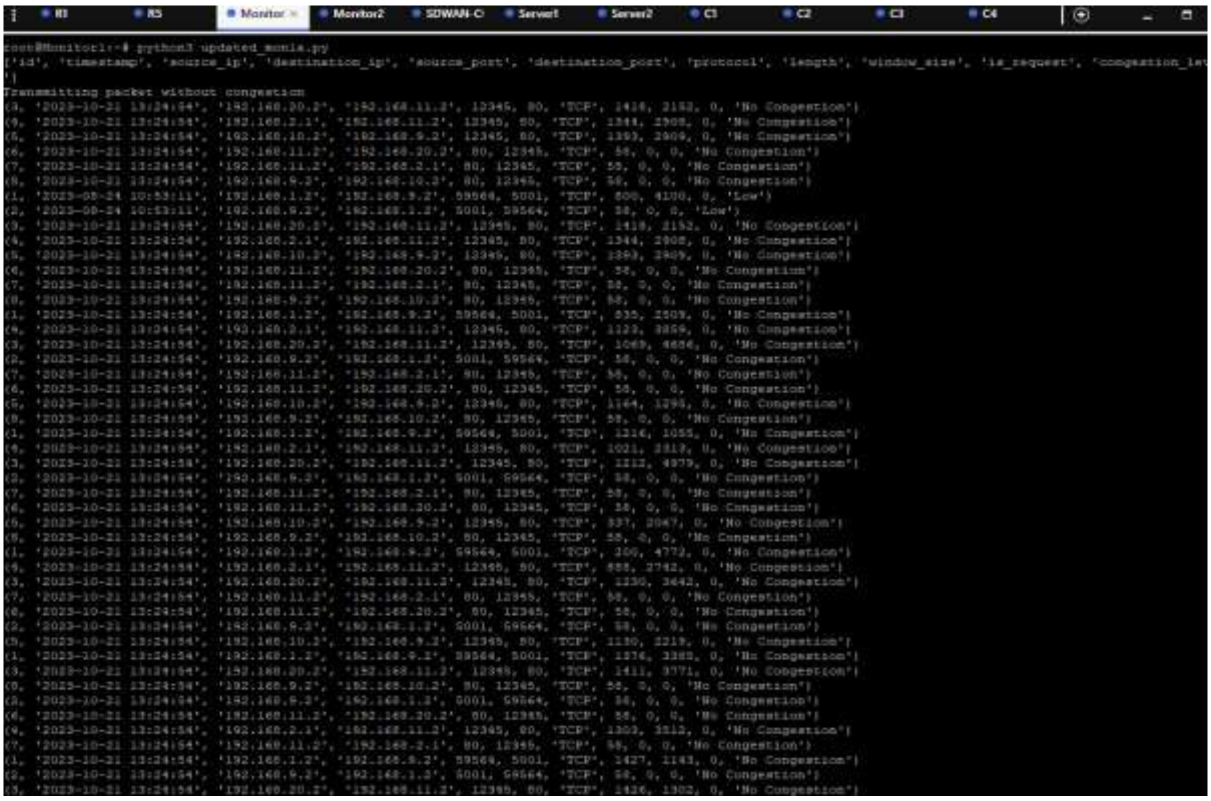
```
root@Monitor1:~# python3 updated_monla.py
['id', 'timestamp', 'source_ip', 'destination_ip', 'source_port', 'destination_port', 'protocol', 'length', 'window_size', 'is_request', '
congestion_level']
Transmitting packet without congestion
(3, '2023-10-21 15:07:23', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1235, 4086, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(2, '2023-08-24 10:53:11', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'Low')
(4, '2023-10-21 15:07:23', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 268, 2515, 0, 'No Congestion')
(5, '2023-10-21 15:07:23', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(6, '2023-10-21 15:07:23', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(7, '2023-10-21 15:07:23', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-08-24 10:53:11', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 800, 4100, 0, 'Low')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(8, '2023-10-21 15:07:23', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 962, 3157, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
Transmitting packets without congestion
(2, '2023-10-21 15:07:23', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 15:07:23', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 15:07:23', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1235, 4086, 0, 'High')
(6, '2023-10-21 15:07:23', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(4, '2023-10-21 15:07:23', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 268, 2515, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:07:24
```

- Functional result for Static Routing Update in the SD-WAN Controller


R1
R5
Monitor1
SDWAN
C1
C2
C3
C4
Server1
Server2
solarwinds | Solar-PuTTY free tool
© 2019 SolarWinds Worldwide

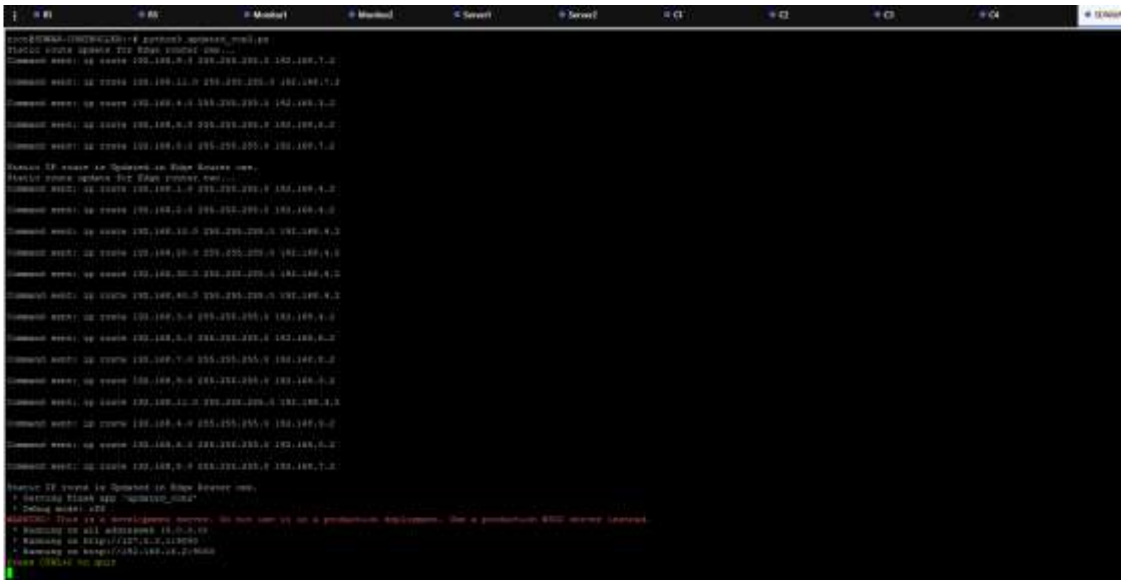
```
root@SDWAN-CONTROLLER:~# python3 updated_con2.py
Static route update for Edge router one...
Command sent: ip route 192.168.9.0 255.255.255.0 192.168.7.2

Command sent: ip route 192.168.11.0 255.255.255.0 192.168.7.2

Command sent: ip route 192.168.4.0 255.255.255.0 192.168.3.2

Command sent: ip route 192.168.6.0 255.255.255.0 192.168.5.2

Command sent: ip route 192.168.8.0 255.255.255.0 192.168.7.2

Static IP route is Updated in Edge Router one.
Static route update for Edge router two...
Command sent: ip route 192.168.1.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.2.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.10.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.20.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.30.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.40.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.3.0 255.255.255.0 192.168.4.2

Command sent: ip route 192.168.5.0 255.255.255.0 192.168.6.2

Command sent: ip route 192.168.7.0 255.255.255.0 192.168.8.2

Command sent: ip route 192.168.9.0 255.255.255.0 192.168.3.2

Command sent: ip route 192.168.11.0 255.255.255.0 192.168.3.2
```

- Functional results showing streams of packet without congestion in the network.



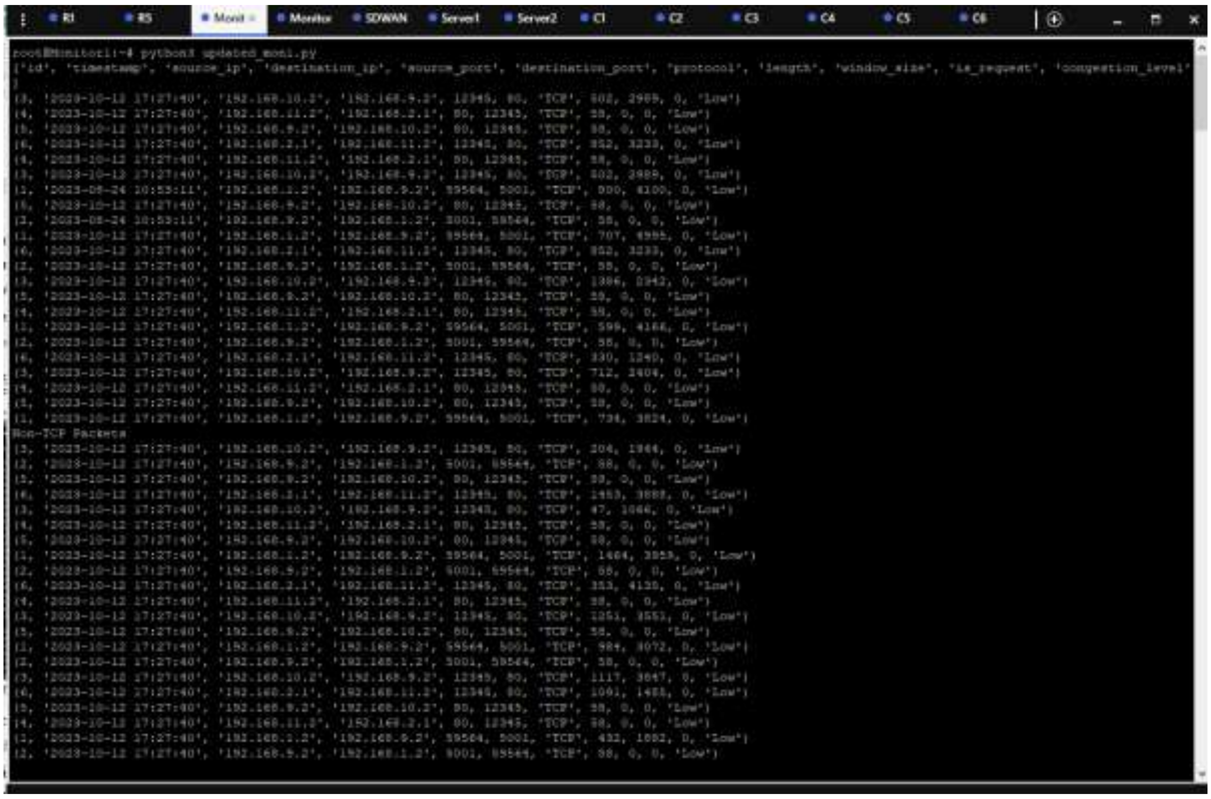
```
root@Monitor1:~# python3 updated_monia.py
['id', 'timestamp', 'source_ip', 'destination_ip', 'source_port', 'destination_port', 'protocol', 'length', 'window_size', 'is_request', 'congestion_lev
']
Transmitting packet without congestion
(3, '2023-10-21 13:24:54', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1416, 2152, 0, 'No Congestion')
(4, '2023-10-21 13:24:54', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1344, 2908, 0, 'No Congestion')
(5, '2023-10-21 13:24:54', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1393, 2909, 0, 'No Congestion')
(6, '2023-10-21 13:24:54', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(7, '2023-10-21 13:24:54', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(8, '2023-10-21 13:24:54', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-08-24 10:53:11', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 800, 4100, 0, 'Low')
(2, '2023-08-24 10:53:11', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'Low')
(3, '2023-10-21 13:24:54', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1416, 2152, 0, 'No Congestion')
(4, '2023-10-21 13:24:54', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1344, 2908, 0, 'No Congestion')
(5, '2023-10-21 13:24:54', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1393, 2909, 0, 'No Congestion')
(6, '2023-10-21 13:24:54', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(7, '2023-10-21 13:24:54', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(8, '2023-10-21 13:24:54', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 13:24:54', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 835, 2509, 0, 'No Congestion')
(4, '2023-10-21 13:24:54', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1123, 3859, 0, 'No Congestion')
(3, '2023-10-21 13:24:54', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1069, 4686, 0, 'No Congestion')
(2, '2023-10-21 13:24:54', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(7, '2023-10-21 13:24:54', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(6, '2023-10-21 13:24:54', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 13:24:54', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1164, 1298, 0, 'No Congestion')
(8, '2023-10-21 13:24:54', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 13:24:54', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 1216, 1055, 0, 'No Congestion')
(4, '2023-10-21 13:24:54', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1021, 2313, 0, 'No Congestion')
(3, '2023-10-21 13:24:54', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1212, 4979, 0, 'No Congestion')
(2, '2023-10-21 13:24:54', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(7, '2023-10-21 13:24:54', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(6, '2023-10-21 13:24:54', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 13:24:54', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 837, 2067, 0, 'No Congestion')
(8, '2023-10-21 13:24:54', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 13:24:54', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 200, 4772, 0, 'No Congestion')
(4, '2023-10-21 13:24:54', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 888, 2742, 0, 'No Congestion')
(3, '2023-10-21 13:24:54', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1230, 3642, 0, 'No Congestion')
(7, '2023-10-21 13:24:54', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(6, '2023-10-21 13:24:54', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(2, '2023-10-21 13:24:54', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 13:24:54', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1130, 2219, 0, 'No Congestion')
(1, '2023-10-21 13:24:54', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 1276, 3385, 0, 'No Congestion')
(3, '2023-10-21 13:24:54', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1411, 3771, 0, 'No Congestion')
(8, '2023-10-21 13:24:54', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(2, '2023-10-21 13:24:54', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(6, '2023-10-21 13:24:54', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(4, '2023-10-21 13:24:54', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1303, 3512, 0, 'No Congestion')
(7, '2023-10-21 13:24:54', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 13:24:54', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 1427, 1143, 0, 'No Congestion')
(2, '2023-10-21 13:24:54', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 13:24:54', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1426, 1302, 0, 'No Congestion')
```

- Functional result of En-SDWAN controller in steady mode because of no congestion



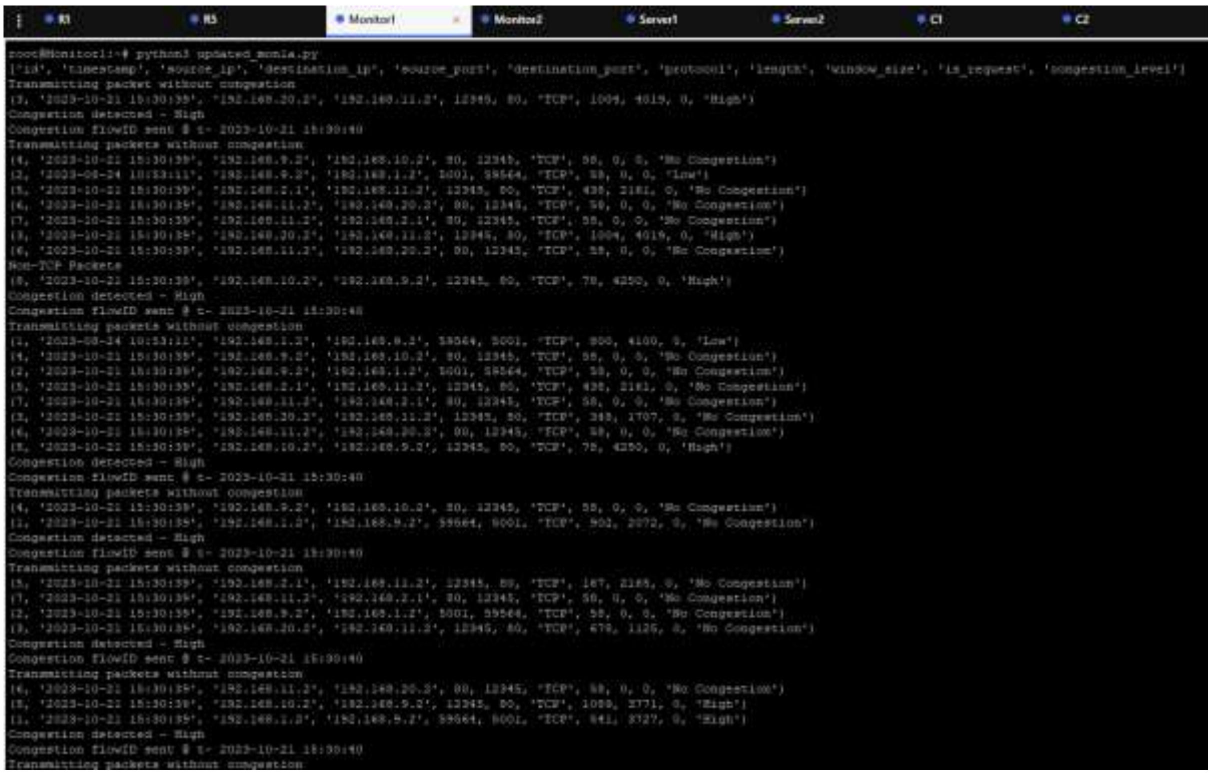
```
root@SDWAN-CONTROLLER:~# python3 updated_[illegible].py
Static route update for Edge router one...
Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Static IP route is Updated in Edge Router one.
Static route update for Edge router two...
Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Command exec: ip route [illegible]

Static IP route is Updated in Edge Router two.
 * Serving Flask app 'updated_[illegible]'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:[illegible]
 * Running on http://192.168.[illegible]
Press CTRL+C to quit
```

- Functional result showing summary of the captured traffic flows in the network architecture.



```
root@Monitor1:~# python3 updated_moni.py
['id', 'timestamp', 'source_ip', 'destination_ip', 'source_port', 'destination_port', 'protocol', 'length', 'window_size', 'is_request', 'congestion_level'
]
(3, '2023-10-12 17:27:40', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 802, 2989, 0, 'Low')
(4, '2023-10-12 17:27:40', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(5, '2023-10-12 17:27:40', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(6, '2023-10-12 17:27:40', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 852, 3233, 0, 'Low')
(4, '2023-10-12 17:27:40', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(3, '2023-10-12 17:27:40', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 802, 2989, 0, 'Low')
(1, '2023-08-24 10:53:11', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 800, 4100, 0, 'Low')
(5, '2023-10-12 17:27:40', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(2, '2023-08-24 10:53:11', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'Low')
(1, '2023-10-12 17:27:40', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 707, 4995, 0, 'Low')
(6, '2023-10-12 17:27:40', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 852, 3233, 0, 'Low')
(2, '2023-10-12 17:27:40', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'Low')
(3, '2023-10-12 17:27:40', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1386, 2942, 0, 'Low')
(5, '2023-10-12 17:27:40', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(4, '2023-10-12 17:27:40', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(1, '2023-10-12 17:27:40', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 599, 4166, 0, 'Low')
(2, '2023-10-12 17:27:40', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'Low')
(6, '2023-10-12 17:27:40', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 330, 1240, 0, 'Low')
(3, '2023-10-12 17:27:40', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 712, 2404, 0, 'Low')
(4, '2023-10-12 17:27:40', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(5, '2023-10-12 17:27:40', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(1, '2023-10-12 17:27:40', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 734, 3824, 0, 'Low')
Non-TCP Packets
(3, '2023-10-12 17:27:40', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 204, 1944, 0, 'Low')
(2, '2023-10-12 17:27:40', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'Low')
(5, '2023-10-12 17:27:40', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(6, '2023-10-12 17:27:40', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1453, 3888, 0, 'Low')
(3, '2023-10-12 17:27:40', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 47, 1066, 0, 'Low')
(4, '2023-10-12 17:27:40', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(5, '2023-10-12 17:27:40', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(1, '2023-10-12 17:27:40', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 1464, 3858, 0, 'Low')
(2, '2023-10-12 17:27:40', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'Low')
(6, '2023-10-12 17:27:40', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 353, 4128, 0, 'Low')
(4, '2023-10-12 17:27:40', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(3, '2023-10-12 17:27:40', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1251, 3551, 0, 'Low')
(5, '2023-10-12 17:27:40', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(1, '2023-10-12 17:27:40', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 984, 3072, 0, 'Low')
(2, '2023-10-12 17:27:40', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'Low')
(3, '2023-10-12 17:27:40', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1117, 3647, 0, 'Low')
(6, '2023-10-12 17:27:40', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 1091, 1488, 0, 'Low')
(5, '2023-10-12 17:27:40', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(4, '2023-10-12 17:27:40', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'Low')
(1, '2023-10-12 17:27:40', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 432, 1882, 0, 'Low')
(2, '2023-10-12 17:27:40', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'Low')
```

- ➢ Functional result showing summary of the captured traffic flows in the network architecture.



```
root@Monitor1:~# python3 updated_monia.py
['id', 'timestamp', 'source_ip', 'destination_ip', 'source_port', 'destination_port', 'protocol', 'length', 'window_size', 'is_request', 'congestion_level']
Transmitting packet without congestion
(3, '2023-10-21 15:30:39', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1004, 4019, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:30:40
Transmitting packets without congestion
(4, '2023-10-21 15:30:39', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(2, '2023-08-24 10:53:11', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'Low')
(5, '2023-10-21 15:30:39', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 438, 2181, 0, 'No Congestion')
(6, '2023-10-21 15:30:39', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(7, '2023-10-21 15:30:39', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 15:30:39', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 1004, 4019, 0, 'High')
(6, '2023-10-21 15:30:39', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
Non-TCP Packets
(8, '2023-10-21 15:30:39', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 78, 4250, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:30:40
Transmitting packets without congestion
(1, '2023-08-24 10:53:11', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 800, 4100, 0, 'Low')
(4, '2023-10-21 15:30:39', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(2, '2023-10-21 15:30:39', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(5, '2023-10-21 15:30:39', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 438, 2181, 0, 'No Congestion')
(7, '2023-10-21 15:30:39', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 15:30:39', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 388, 1707, 0, 'No Congestion')
(6, '2023-10-21 15:30:39', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(8, '2023-10-21 15:30:39', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 78, 4250, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:30:40
Transmitting packets without congestion
(4, '2023-10-21 15:30:39', '192.168.9.2', '192.168.10.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(1, '2023-10-21 15:30:39', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 902, 2072, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:30:40
Transmitting packets without congestion
(5, '2023-10-21 15:30:39', '192.168.2.1', '192.168.11.2', 12345, 80, 'TCP', 187, 2185, 0, 'No Congestion')
(7, '2023-10-21 15:30:39', '192.168.11.2', '192.168.2.1', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(2, '2023-10-21 15:30:39', '192.168.9.2', '192.168.1.2', 5001, 59564, 'TCP', 58, 0, 0, 'No Congestion')
(3, '2023-10-21 15:30:39', '192.168.20.2', '192.168.11.2', 12345, 80, 'TCP', 678, 1126, 0, 'No Congestion')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:30:40
Transmitting packets without congestion
(6, '2023-10-21 15:30:39', '192.168.11.2', '192.168.20.2', 80, 12345, 'TCP', 58, 0, 0, 'No Congestion')
(8, '2023-10-21 15:30:39', '192.168.10.2', '192.168.9.2', 12345, 80, 'TCP', 1088, 2771, 0, 'High')
(1, '2023-10-21 15:30:39', '192.168.1.2', '192.168.9.2', 59564, 5001, 'TCP', 841, 3727, 0, 'High')
Congestion detected - High
Congestion flowID sent @ t- 2023-10-21 15:30:40
Transmitting packets without congestion
```

REFERENCES

Abraham, E. (2023). *Find the Perfect SD-WAN Solution for Your Needs*. The Watch Guard Networks. Retrieved 18-04-2023 from https://www.watchguard.com/wgrd-solutions/security-topics/sd-wan

Aditya, S. (2020). *Software Defined Overlay Network and Workload Orchestration in Social Network Based Edge Infrastructure for Smart Communities* University of Florida].

Agarwal, V., Saraswat, S., Gupta, H. P., & Dutta, T. (2018). A traffic engineering framework for maximizing network revenue in software defined network. 2018 10th International Conference on Communication Systems & Networks (COMSNETS),

Ahmed, H. G., & Ramalakshmi, R. (2018). Performance analysis of centralized and distributed SDN controllers for load balancing application. 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI),

Al-Haddad, R., & Velazquez, E. S. (2019). A Survey of Quality of Service (QoS) Protocols and Software-Defined Networks (SDN) From the Traditional to the Latest Network Architecture. Intelligent Computing: Proceedings of the 2018 Computing Conference, Volume 2,

Al-Saadi, R., Armitage, G., But, J., & Branch, P. (2019). A survey of delay-based and hybrid TCP congestion control algorithms. *IEEE Communications Surveys & Tutorials*, *21*(4), 3609-3638.

Al-Sadi, A. (2018). *A Novel Placement Algorithm for the Controllers Of the Virtual Networks (COVN) in SD-WAN with Multiple VNs* University of Northampton].

Aldossary, M. (2021). A Review of Dynamic Resource Management in Cloud Computing Environments. *Comput. Syst. Sci. Eng.*, *36*(3), 461-476.

Ali, J., Lee, G.-M., Roh, B.-H., Ryu, D. K., & Park, G. (2020). Software-defined networking approaches for link failure recovery: A survey. *Sustainability*, *12*(10), 4255.

Allman, M., Paxson, V., & Blanton, E. (2009). *TCP congestion control* (2070-1721).

Altalebi, O. W. J., & Ibrahim, A. A. (2022). Optimization of Elapsed Time of Automation for Large-Scale Traditional Networks and Proposing New Automation Scripts. 2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA),

Amiri, E., Alizadeh, E., & Rezvani, M. H. (2021). Optimized controller placement for software defined wide area networks. 2021 7th International Conference on Web Research (ICWR),

Anderson, M. (2023). *Software Defined Wide-Area Networking: Four must do things to get Right* The Accenture Consulting. https://www.accenture.com/_acnmedia/PDF-88/Accenture-Designed-SDN-POV-Review.pdf#zoom=50

Aydeger, A., Akkaya, K., & Uluagac, A. S. (2015). SDN-based resilience for smart grid communications. 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN),

Bannour, F., Souihi, S., & Mellouk, A. (2017). Distributed SDN control: Survey, taxonomy, and challenges. *IEEE Communications Surveys & Tutorials*, *20*(1), 333-354.

Bar-Lev, D. (2022). *SD-WAN Controller*. MEF Forum. Retrieved Oct 22, 2020 from https://wiki.mef.net/display/CESG/SD-WAN+Controller

Barracuda, M. (2023). *Barracuda Load Balancer ADC Secure Application Delivery & Load Balancing. The Barracuda Networks*. 2023. https://www.barracuda.com/products/application-protection/load-balancer/solutions/rds

Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., & Snow, W. (2014). ONOS: towards an open, distributed SDN OS. Proceedings of the third workshop on Hot topics in software defined networking, Berger, G. (2023). *NODEFLOW: An openflow controller node style*. The nodeflow Retrieved 06-8-2023 from https://cranky-tesla-91fbb5.netlify.app/

Bigelow, C. L. S. J. (2023). *IT infrastructure*. Tech Target Network. https://www.techtarget.com/searchdatacenter/definition/infrastructure

Bing, B. (2008). Emerging technologies in wireless LANs: theory, design, and deployment.

Blidborg, E. (2022). An Overview of Monitoring Challenges That Arise With SD-WAN.

Braun, W., & Menth, M. (2014). Software-defined networking using OpenFlow: Protocols, applications and architectural design choices. *Future Internet*, *6*(2), 302-336.

Cai, N., Han, Y., Ben, Y., An, W., & Xu, Z. (2019). An effective load balanced controller placement approach in software-defined WANs. MILCOM 2019-2019 IEEE Military Communications Conference (MILCOM),

Cainkar, P. (2023). *Failsafe Reliability and Exceptional Quality of Experience for a Multicloud World: The Oracle Cloud Networks.* The Oracle Cloud Networks. Retrieved 14-04-2023 from https://www.oracle.com/cloud/networking/

Carvajal, J. M., Gilabert, F. T., & Cañadas, J. (2021). Corporate network transformation with SD-WAN. A practical approach. 2021 Eighth International Conference on Software Defined Systems (SDS),

Cimorelli, F. (2018). SDN Workload Balancing and QoE Control in Next Generation Network Infrastructures.

Cisco, U. (2020). Cisco annual internet report (2018–2023) white paper. *Cisco: San Jose, CA, USA*, *10*(1), 1-35.

Cole, Z. (2023). *Network simulation or emulation?* The NetworkWorld. Retrieved 12-9-2023 from https://www.networkworld.com/article/3227076/network-simulation-or-emulation.html

Combs, H. (2023). *Wireshark*. The Wireshark. Retrieved 20-9-2023 from https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroWhatIs

Cooney, M. (2023a). *What is SD-WAN, and what does it mean for networking, security, cloud?* The Networkworld United Kingdom. Retrieved 20 OCTOBER 2022 from https://www.networkworld.com/article/3031279/sd-wan-what-it-is-and-why-you-ll-use-it-one-day.html

Cooney, M. (2023b). *What is SDN and where software-defined networking is going*. The NetworkWorld. Retrieved 2023 from https://www.networkworld.com/article/3209131/what-sdn-is-and-where-its-going.html

Corn, M. (2023). *MEF 3.0 Workshop - MEF 3.0 SD-WAN Certification*. The Metro Ethernet Forum. Retrieved 18 November 2019 from https://www.youtube.com/watch?v=1pX47FGt_zY

Creta, G. (2018). *What is Software-Defined Wide Area Networking (or SD-WAN/SDWAN) and how can it help your business?* FlexNetworks. Retrieved 22-08-2023 from https://flexnetworks.ca/what-is-sd-wan-how-help-business/

Dante, D. (2023). *Secure Enterprise SD-WAN: Shift into the Fast Lane*. The Riverbed. Retrieved 20-04-2023 from https://www.riverbed.com/solutions/sd-wan-software-defined-wan/

Davie, B., Koponen, T., Pettit, J., Pfaff, B., Casado, M., Gude, N., Padmanabhan, A., Petty, T., Duda, K., & Chanda, A. (2017). A database approach to sdn control plane design. *ACM SIGCOMM Computer Communication Review*, *47*(1), 15-26.

Davies, B. (2022). Eight (8) Load Balancing techniques you should know. https://lavellenetworks.com/blog/8-load-balancing-techniques-you-should-know/

Deepa, T., & Cheelu, D. (2017). A comparative study of static and dynamic load balancing algorithms in cloud computing. 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), Digg. (2023). *Cisco's PPDIOO Network Cycle*. Cisco. Retrieved 6-8-2023 from https://www.ciscopress.com/articles/article.asp?p=1697888

Dooley, K. (2023). *What Is a Network Diagram?* Auvik's Network Retrieved 8-9-2023 from https://www.auvik.com/franklyit/blog/network-diagrams-explained/

Duliński, Z., Stankiewicz, R., Rzym, G., & Wydrych, P. (2020). Dynamic traffic management for sd-wan inter-cloud communication. *IEEE Journal on Selected Areas in Communications*, *38*(7), 1335-1351.

Durand, A., Droms, R., Woodyatt, J., & Lee, Y. (2011). RFC 6333: Dual-stack lite broadband deployments following IPv4 exhaustion. In: RFC Editor.

Ebuah, C. s. (2023). *Partner Technology Solutions. Cradle Points Networks.* The Cradle Points Networks. Retrieved 15-04-2023 from https://cradlepoint.com/partners/for-customers/partner-technology-solutions/

Edwards, J. (2023). *SD-WAN's Benefits Extend Beyond Cost Savings*. The Network Computing Newsletter. Retrieved MARCH 27, 2019 from https://www.networkcomputing.com/networking/sd-wans-benefits-extend-beyond-cost-savings

Eissa, H. A., Bozed, K. A., & Younis, H. (2019). Software defined networking. 2019 19th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA),

Erickson, D. (2013). The beacon openflow controller. Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking,

Fajjari, I., Aitsaadi, N., & Kouicem, D. E. (2017). A novel SDN scheme for QoS path allocation in wide area networks. GLOBECOM 2017-2017 IEEE Global Communications Conference,

Farhady, H., Lee, H., & Nakao, A. (2015). Software-defined networking: A survey. *Computer Networks*, *81*, 79-95.

FatPipe-Networks. (2023). *Software-Defined Networking (SD-WAN) for Hybrid Multi-Line Wide -Area Networks with High Security*. The FatPipe Networks. Retrieved 15-05-2023 from https://www.fatpipeinc.com/

Feamster, N., Rexford, J., & Zegura, E. (2014). The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, *44*(2), 87-98.

Flexiwan, Z. (2023). *The World's First Open Source SD-WAN & SASE*. The Flexiwan Online resources. Retrieved 10-04-2023 from https://flexiwan.com/

Floyd, S., Henderson, T., & Gurtov, A. (2004). *The NewReno modification to TCP's fast recovery algorithm* (2070-1721).

Franjić, V. (2023). *Cisco SD-WAN: Delivering Cisco Next Generation SD-WAN with Viptela. The Cisco Viptela Networks*. The Cisco Enterprise. Retrieved 05-05-2023 from https://www.cisco.com/c/dam/m/hr_hr/training-events/2019/cisco-connect/pdf/delivering_cisco_next_generation_sd-wan_with_viptela_vedran_franjic.pdf

Fruhlinger, J. (2023). *What is SDN and where is it going?* The Network world. Retrieved 24 of August 2023 from https://www.networkworld.com/article/3209131/what-sdn-is-and-where-its-going.html

Ghode, R. (2023). *Software Defined Wide Area Networks Market Share*. The Markets and Markets. Retrieved 05-01-2023 from https://www.marketsandmarkets.com/Market-Reports/software-defined-wan-market-53110642.html#:~:text=%5B263%20Pages%20Report%5D%20The%20global,the%20SD%2DWAN%20market%20growth.

Ghodsi, A., Shenker, S., Koponen, T., Singla, A., Raghavan, B., & Wilcox, J. (2011). Intelligent design enables architectural evolution. Proceedings of the 10th ACM Workshop on hot topics in networks, Gkioulos, V., Gunleifsen, H., & Weldehawaryat, G. K. (2018). A systematic literature review on military software defined networks. *Future Internet*, *10*(9), 88.

Golani, K., Goswami, K., Bhatt, K., & Park, Y. (2018). Fault tolerant traffic engineering in software-defined WAN. 2018 IEEE Symposium on Computers and Communications (ISCC),

Goransson, P., Black, C., & Culver, T. (2016). *Software defined networks: a comprehensive approach*. Morgan Kaufmann.

Granath, D. (2023). *What is SD-WAN?* The Aruba Networks. https://www.arubanetworks.com/faq/what-is-sd-wan

Greene, N. (2023). *Delivering Digital Transformation At Scale:*

Network Trends and Architectures. https://cdn.idc.com/downloads/idc_interop_2016_final.pdf

Grgurevic, I., Barišić, G., & Stančić, A. (2021). Analysis of MPLS and SD-WAN Network Performances Using GNS3. International Conference on Future Access Enablers of Ubiquitous and Intelligent Infrastructures,

Guardabaxo, H. H. G., & Pavani, G. S. (2022). QoE Management of HTTP/2 traffic in Software-Defined Wide Area Networks. NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium,

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., & Shenker, S. (2008). NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, *38*(3), 105-110.

Guo, Z., Dou, S., Liu, S., Feng, W., Jiang, W., Xu, Y., & Zhang, Z.-L. (2022). Maintaining control resiliency and flow programmability in software-defined wans during controller failures. *IEEE/ACM Transactions on Networking*, *30*(3), 969-984.

Guo, Z., Dou, S., Wang, Y., Liu, S., Feng, W., & Xu, Y. (2021). HybridFlow: Achieving load balancing in software-defined WANs with scalable routing. *IEEE Transactions on Communications*, *69*(8), 5255-5268.

Ha, S., Rhee, I., & Xu, L. (2008). CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, *42*(5), 64-74.

Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D. C., & Gayraud, T. (2014). Software-defined networking: Challenges and research opportunities for future internet. *Computer Networks*, *75*, 453-471.

Hamdan, O. A., Strachan, S., & Arslan, E. (2022). Bandwidth and Congestion Aware Routing for Wide-Area Hybrid Networks. 2022 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN),

Hartert, R., Vissicchio, S., Schaus, P., Bonaventure, O., Filsfils, C., Telkamp, T., & Francois, P. (2015). A declarative and expressive approach to control forwarding paths in carrier-grade networks. *ACM SIGCOMM Computer Communication Review*, *45*(4), 15-28.

Hassas Yeganeh, S., & Ganjali, Y. (2012). Kandoo: a framework for efficient and scalable offloading of control applications. Proceedings of the first workshop on Hot topics in software defined networks,

Heller, B., Sherwood, R., & McKeown, N. (2012). The controller placement problem. *ACM SIGCOMM Computer Communication Review*, *42*(4), 473-478.

Heng, D. (2018). *What Is an Overlay Network?* Huawei. Retrieved 29-01-2024 from https://info.support.huawei.com/info-finder/encyclopedia/en/Overlay+network.html

Hou, X., Muqing, W., Bo, L., & Yifeng, L. (2019). Multi-controller deployment algorithm in hierarchical architecture for SDWAN. *IEEE Access*, *7*, 65839-65851.

Hu, F., Hao, Q., & Bao, K. (2014). A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys & Tutorials*, *16*(4), 2181-2206.

Hu, X., Xiang, Y., Li, Y., Qiu, B., Wang, K., & Li, J. (2021). Trident: Efficient and practical software network monitoring. *Tsinghua Science and Technology*, *26*(4), 452-463.

Huang, H.-M., Tidwell, T., Gill, C., Lu, C., Gao, X., & Dyke, S. (2010). Cyber-physical systems for real-time hybrid structural testing: a case study. Proceedings of the 1st ACM/IEEE international conference on cyber-physical systems,

IBM. (2023). *Top five challenges of managing SD-WAN and wifi in the modern branch office*. IBM. https://www.ibm.com/downloads/cas/QXZZOMEA

Ibrahim Hussein, S. A., Zaki, F. W., & Ashour, M. M. (2022). Performance evaluation of software-defined wide area network based on queueing theory. *IET Networks*, *11*(3-4), 128-145.

Ivanisenko, I. (2015). Methods and Algorithms of load balancing. *International Journal Information Technologies & Knowledge*, *9*(4), 340-375.

Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., & Zhu, M. (2013). B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review*, *43*(4), 3-14.

Jamjoom, H., Williams, D., & Sharma, U. (2014). Don't call them middleboxes, call them middlepipes. Proceedings of the third workshop on Hot topics in software defined networking,

Jammal, M., Singh, T., Shami, A., Asal, R., & Li, Y. (2014). Software defined networking: State of the art and research challenges. *Computer Networks*, *72*, 74-98.

Jay Schulist, D. B., Alexei Starovoitov. (2023). *Linux Socket Filtering aka Berkeley Packet Filter (BPF)*. The Kernel. Retrieved 12-9-2023 from https://www.kernel.org/doc/html/v5.10/networking/filter.html

Ji, J. (2023). *RabbitMQ is the most widely deployed open source message broker*. https://www.rabbitmq.com/

Joy, S. (2023). *SDWAN Congestion Management*. LinkedIn. Retrieved July 16, 2020 from https://www.linkedin.com/pulse/sdwan-congestion-management-joy-sarkar/?trk=public_profile_article_view

Juniper-Networks. (2023a). Contrail Service Orchestration CSO SD-WAN – Design and Architecture Guide. https://www.juniper.net/documentation/us/en/software/cso/sg-007-sd-wan-sd-lan-design-arch-guide/sg-007-sd-wan-sd-lan-design-arch-guide.pdf

Juniper-Networks. (2023b). *Zero Touch Provisioning*. Juniper Networks https://www.juniper.net/documentation/us/en/software/junos/junos-install-upgrade/topics/topic-map/zero-touch-provision.html

Kannan, K. (2023). *Nokia Software Defined Access Networks to build smarter networks*. The Nokia Telecommunication Retrieved 20 of August 2023 from https://www.nokia.com/about-us/news/releases/2017/10/16/du-trials-nokia-software-defined-access-networks-to-build-smarter-networks/

Karakus, M., & Durresi, A. (2017). A survey: Control plane scalability issues and approaches in software-defined networking (SDN). *Computer Networks*, *112*, 279-293.

Kazarez, A. (2023). Loom Github Page. In.

Keary, T. (2020). *Software-Defined Networking – SDN Guide*. Comparing Technology Retrieved 20 from https://www.comparitech.com/net-admin/software-defined-networking/

Kharub, J. (2022). Technical Analysis of Various Vendor SD-WAN Offering.

Khoury, J. (2023). *Juniper SD-WAN driven by Mist AI Enrich user experiences across the WAN with AI-driven insight, automation, and action. The Jupiter Networks*. The Jupiter Networks. Retrieved 10-05-2023 from https://www.juniper.net/uk/en/solutions/sd-wan/

Kim, H., & Feamster, N. (2013). Improving network management with software defined networking. *IEEE Communications Magazine*, *51*(2), 114-119.

Kiran, S. (2023). *Managed SD-WAN Solutions for the Cloud Era*. The Aryaka Group of Company. https://www.aryaka.com/managed-wan-services/

Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., & Hama, T. (2010). Onix: A distributed control platform for large-scale production networks. OSDI,

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2014). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, *103*(1), 14-76.

Kuzmanovic, A., & Knightly, E. W. (2006). TCP-LP: low-priority service via end-point congestion control. *IEEE/ACM Transactions on Networking*, *14*(4), 739-752.

Laliberte, B. (2023). *The Importance of Network Visibility and Analytics for Zero*

Trust Initiatives. The Fortinet (Fortigate). Retrieved 15-02-2023 from https://www.fortinet.com/content/dam/fortinet/assets/analyst-reports/esg-importance-network-visibility.pdf

Landsberger, D. (2023). *SDN, SD-WAN, NFV, VNF: What Are the Differences, and Why Should You Care?* The Comptia. Retrieved December 17, 2021 from https://www.comptia.org/blog/sdn-sd-wan-nfv-vnf

Latif, Z., Sharif, K., Li, F., Karim, M. M., Biswas, S., & Wang, Y. (2020). A comprehensive survey of interface protocols for software defined networks. *Journal of Network and Computer Applications*, *156*, 102563.

Lee, D. (2023). *Benefits of a SD-WAN Development Ecosystem. The CloudGenix Networks*. The CloudGenix Networks. Retrieved 10-04-2023 from https://sd-wan.cloudgenix.com/rs/911-KCN-503/images/CloudGenix_Analyst_Report_Doyle.pdf

Lemeshko, O., Yeremenko, O., Yevdokymenko, M., Zhuravlova, A., Kruhlova, A., & Lemeshko, V. (2021). Research of improved traffic engineering fault-tolerant routing mechanism in SD-WAN. 2021 IEEE 3rd Ukraine Conference on Electrical and Computer Engineering (UKRCON),

Lerner, A. (2018). *Checking in on the Death of the CLI*. The Gartner. Retrieved January 04, 2023 from https://blogs.gartner.com/andrew-lerner/2018/01/04/checking-in-on-the-death-of-the-cli/

Lerner, A. (2023). *Look Beyond Network Vendors for Network Innovation*. Retrieved 23 January 2018 from https://www.gartner.com/en/documents/3847469

Li, S., Hu, D., Fang, W., Ma, S., Chen, C., Huang, H., & Zhu, Z. (2017). Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability. *IEEE Network*, *31*(2), 58-66.

Liang, G., & Li, W. (2018). A novel industrial control architecture based on Software-Defined Network. *Measurement and Control*, *51*(7-8), 360-367.

Lin, S.-C., Wang, P., & Luo, M. (2016). Control traffic balancing in software defined networks. *Computer Networks*, *106*, 260-271.

Liu, S., & Li, B. (2015). On scaling software-defined networking in wide-area networks. *Tsinghua Science and Technology*, *20*(3), 221-232.

Manel, M., & Habib, Y. (2017). An efficient MPLS-based source routing scheme in software-defined wide area networks (SD-WAN). 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA),

Manova, R. Y., Sukmadirana, E., & Nurmanah, N. S. (2022). Comparative Analysis of Quality of Service and Performance of MPLS, EoIP and SD-WAN. 2022 1st International Conference on Information System & Information Technology (ICISIT),

Masoudi, R., & Ghaffari, A. (2016). Software defined networks: A survey. *Journal of Network and Computer Applications*, *67*, 1-25.

Maswood, M. M. S., Rahman, M. R., Alharbi, A. G., & Medhi, D. (2020). A novel strategy to achieve bandwidth cost reduction and load balancing in a cooperative three-layer fog-cloud computing environment. *IEEE Access*, *8*, 113737-113750.

Mazin, A. M., Ab Rahman, R., & Kassim, M. (2021). Performance analysis on network automation interaction with network devices using python. 2021 IEEE 11th IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE),

McCanne, V. J. C. L. S. (2023). *TCP Dump and PCAP*. The TCPDump Organisation. Retrieved 10-9-2023 from https://www.tcpdump.org/manpages/pcap.3pcap.html#lbAS

MEF88. (2023). *MEF 88 - Application Flow Security for SD-WAN Services* The Metro Ethernet Forum. Retrieved 25 May 2023 from https://www.mef.net/wp-content/uploads/MEF_88.pdf

Mehra, R. (2023). SD-WAN: Security, Application Experience and Operational Simplicity Drive Market Growth. Retrieved 15 - 6 - 2023, from

Mena Diaz, C. J., Utrilla Arellano, J. G., Andrade Arenas, L., & Cano Lengua, M. A. (2022). Analysis about benefits of software-defined wide area network: a new alternative for WAN connectivity.

Mittal, R., Lam, V. T., Dukkipati, N., Blem, E., Wassel, H., Ghobadi, M., Vahdat, A., Wang, Y., Wetherall, D., & Zats, D. (2015). TIMELY: RTT-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review*, *45*(4), 537-550.

Mohammadi, A. A., Hussain, R., Oracevic, A., Kazmi, S. M. A. R., Hussain, F., Aloqaily, M., & Son, J. (2022). A Novel TCP/IP Header Hijacking Attack on SDN. IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS),

Mohd Fuzi, M. F., Abdullah, K., Abd Halim, I. H., & Ruslan, R. (2021). Network automation using ansible for EIGRP network. *Journal of Computing Research and Innovation (JCRINN)*, *6*(4), 59-69.

Montazerolghaem, A., Moghaddam, M. H. Y., & Leon-Garcia, A. (2017). OpenSIP: Toward software-defined SIP networking. *IEEE transactions on network and service management*, *15*(1), 184-199.

Mora-Huiracocha, R. E., Gallegos-Segovia, P. L., Vintimilla-Tapia, P. E., Bravo-Torres, J. F., Cedillo-Elias, E. J., & Larios-Rosillo, V. M. (2019). Implementation of a SD-WAN for the interconnection of two software defined data centers. 2019 IEEE Colombian Conference on Communications and Computing (COLCOM),

Morales, L. V., Murillo, A. F., & Rueda, S. J. (2015). Extending the floodlight controller. 2015 IEEE 14th International Symposium on Network Computing and Applications,

Nangare, S. (2023). *SD-WAN Market Statistics, Predictions, Case Studies*. The International Data Corporation. Retrieved 5th May from https://www.slideshare.net/SagarNangare/sdwan-market-statistics-predictions-case-stuties

Navarro, A., Canonico, R., & Botta, A. (2023). Software Defined Wide Area Networks: Current Challenges and Future Perspectives. 2023 IEEE 9th International Conference on Network Softwarization (NetSoft),

NetBrain. (2023). *Network Automation*. Delta Line International. Retrieved 07 from https://deltaline-it.com/solutions/network-automation

Network-Academy. (2023). *Underlay vs Overlay Routing*. The Network Academy Retrieved 12-08-2023 from https://www.networkacademy.io/ccie-enterprise/sdwan/underlay-vs-overlay-routing

Newberg, J.-m. (2023). *GNS3 comptible switch that has a functionl port mirroring ability*. GNS3 Retrieved 10-10-2023 from https://gns3.com/community/featured/gns3-comptible-switch-that-has-a

Nicholas, J. (2023). SD-WAN / SD-Branch Architecture for Enterprise. https://fortinetweb.s3.amazonaws.com/docs.fortinet.com/v2/attachments/7030e0d2-4287-11ec-bdf2-fa163e15d75b/secure-sdwan-7.0-arch-for-enterprise.pdf

Norling, L. (2023). *SD-WAN as a Service*. The Catonetworks. Retrieved 05-05-2023 from https://www.catonetworks.com/sd-wan/sd-wan-as-a-service/

NRaghava, N., & Singh, D. (2014). Comparative study on load balancing techniques in cloud computing. *Open journal of mobile computing and cloud computing*, *1*(1), 31-42.

Nuage-Networks. (2023). *SD-WAN 2.0 for Enterprise Networks*. The Nuage Networks. Retrieved 10-05-2023 from https://www.nuagenetworks.net/solutions/software-defined-wan/

Ominike Akpovi, A., Adebayo, A., & Osisanwo, F. (2016). Introduction to Software Defined Networks (SDN).. *International Journal of Applied Information Systems*, 10-14.

ONF. (2023). *Metro Ethernet Forum (MEF)*. The Open Network Foundation Retrieved 15 August 2023 from https://opennetworking.org/about-onf/liaisons/mef/

Orans, L., D'Hoinne, J., & Chessman, J. (2020). Market Guide for Network Detection and Response. In: Retrieved from Gartner: www. gartner. com.

PaloAlto-Networks. (2023). *PAN-OS Web Interface Help SD-WAN Monitoring*. P. A. N. Publications. https://docs.paloaltonetworks.com/pan-os/10-2/pan-os-web-interface-help/panorama-web-interface/panorama-sd-wan/sd-wan-monitoring

Papavassiliou, S. (2020). Software defined networking (SDN) and network function virtualization (NFV). In (Vol. 12, pp. 7): MDPI.

Parent, F. (2023). *Business Network Solutions: Agile and secure cloud connectivity*. The Adapative Networks. Retrieved 03-05-2023 from https://www.adaptiv-networks.com/

Parra, G. D. L. T., Rad, P., & Choo, K.-K. R. (2019). Implementation of deep packet inspection in smart grids and industrial Internet of Things: Challenges and opportunities. *Journal of Network and Computer Applications*, *135*, 32-46.

Pashkov, V. (2022). Design of Highly Available Distributed Control Plane for Software-Defined Networks. 2022 International Conference on Modern Network Technologies (MoNeTec),

Patel, P., Tiwari, V., & Abhishek, M. K. (2016). SDN and NFV integration in openstack cloud to improve network services and security. 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT),

Peterson, L., Cascone, C., & Davie, B. (2021). *Software-defined networks: a systems approach*. Systems Approach, LLC.

Petryschuk, S. (2023). *Network Design and Best Practices*. Auvik Networks. Retrieved 09-09-2023 from https://www.auvik.com/franklyit/blog/network-design-best-practices/

Phemius, K., Bouet, M., & Leguay, J. (2014). Disco: Distributed multi-domain sdn controllers. 2014 IEEE network operations and management symposium (NOMS),

Piech, M. (2023). *Red Hat Takes the Guesswork out of Cloud, Hybrid and Internet-of-Things Integration*. The Red Hat https://www.redhat.com/de/about/press-releases/red-hat-takes-guesswork-out-cloud-hybrid-and-internet-things-integration

Pollock, H. (2023). *The Citrix SD-WAN Product Documentation. The Citrix Networks*. The Citrix Networks. Retrieved 24-08-2022 from https://docs.netscaler.com/en-us/citrix-sd-wan.html

Power, M. (2023). *MEF Introduces New Standards for High-Performance, Secure SD-WAN Services*. The Metro Ethernet Forum Retrieved 15 June 2023 from https://www.mef.net/news/mef-introduces-new-standards-for-high-performance-secure-sd-wan-services/?utm_source=TelecomTV&utm_campaign=d3c685149b-EMAIL_CAMPAIGN_2021_12_19_08_34&utm_medium=email&utm_term=0_6197c572c4-d3c685149b-166678921

Q-Balancer. (2023). Solution Brief: Multi-WAN Load Balancing for Enterprises. http://www.q-balancer.com/phocadownload/User-Documentation/Solution-Brief_Multi-WAN-Load-Balancing-for-Enterprises.pdf

Qiu, K., Huang, S., Xu, Q., Zhao, J., Wang, X., & Secci, S. (2017). Paracon: A parallel control plane for scaling up path computation in sdn. *IEEE transactions on network and service management*, *14*(4), 978-990.

Raghunathan, D. N. S. (2021). *SD-WAN: The Key Enabler for Network Transformation with Unmatched Benefits*. C. Corporation. https://www.cisco.com/c/dam/global/en_in/assets/pdfs/idc-customers-sd-wan.pdf

Rajagopalan, S. (2020). An Overview of SD-WAN Load Balancing for WAN Connections. 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA),

Raynovich, S. (2023). *THE 2021 SD-WAN GROWTH REPORT IS HERE!* The Futuriom. Retrieved 01-05-2023 from https://www.futuriom.com/articles/news/the-2021-sd-wan-growth-report-is-here/2021/06?utm_source=TelecomTV&utm_campaign=c6f9ffe5ee-EMAIL_CAMPAIGN_2021_12_09_08_09&utm_medium=email&utm_term=0_6197c572c4-c6f9ffe5ee-166678921

Rohyans, A., Shaikh, A., Rajaram, C., Klebanov, D., Kumar, D., Cornett, G., Malik, H., Ghodgaonkar, K., Arunachalam, M., & Pitaev, N. (2019). Cisco SD-WAN Cloud scale architecture. *Cisco,[Online]. Available: https://www. cisco. com/c/dam/en/us/solutions/collateral/enterprise-networks/sd-wan/nb-06-cisco-sd-wan-ebook-cte-en. pdf*.

Ryu, S. (2022). Framework Community Ryu Controller. In.

Sabiq, A. T., Karimah, S. A., & Jadied, E. M. (2023). Analisis Perbandingan UDP dan DCCP Pada Jaringan SD-WAN. *eProceedings of Engineering*, *10*(3).

Sahoo, K. S., Mishra, P., Tiwary, M., Ramasubbareddy, S., Balusamy, B., & Gandomi, A. H. (2019). Improving end-users utility in software-defined wide area network systems. *IEEE transactions on network and service management*, *17*(2), 696-707.

Sainz, M., Garitano, I., Iturbe, M., & Zurutuza, U. (2020). Deep packet inspection for intelligent intrusion detection in software-defined industrial networks: A proof of concept. *Logic Journal of the IGPL*, *28*(4), 461-472.

Salisbury, B. (2023). *OpenFlow: Proactive vs Reactive Flows*. The Network Static. Retrieved 10-9-2023 from http://networkstatic.net/openflow-proactive-vs-reactive-flows/

Scarpati, J. (2023). *Network hub*. Tech Target. Retrieved 12-9-2023 from https://www.techtarget.com/searchnetworking/definition/hub

Scarpitta, C., Ventre, P. L., Lombardo, F., Salsano, S., & Blefari-Melazzi, N. (2021). EveryWAN-an open source SD-WAN solution. 2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME),

Septyanto, R. B., & Ikasari, D. (2021). Data Traffic Performance Analysis of SD-WAN Network Technology Using Tableau Software. *Budapest International Research and Critics Institute-Journal (BIRCI-Journal)*, *4*(3), 6341-6356.

Šeremet, I., & Čaušević, S. (2019). Advancing ip/impls with software defined network in wide area network. 2019 International Workshop on Fiber Optics in Access Networks (FOAN),

Sezer, S., Scott-Hayward, S., Chouhan, P. K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., & Rao, N. (2013). Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Communications Magazine*, *51*(7), 36-43.

Shah, N. (2023). *Do Not Underestimate the Challenge of Securing SD-WAN*. The Fortinet Technology. Retrieved February 04, 2019 from https://www.fortinet.com/blog/business-and-technology/do-not-underestimate-the-challenge-of-securing-sd-wan

Shalunov, S., Hazel, G., Iyengar, J., & Kuehlewind, M. (2012). *Low extra delay background transport (LEDBAT)* (2070-1721).

Shaw, K. (2023). *What is a network switch, and how does it work?* The NetworkWorld Retrieved 10-9-2023 from https://www.networkworld.com/article/3584876/what-is-a-network-switch-and-how-does-it-work.html

Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., & Parulkar, G. (2009). Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, *1*, 132.

Shozi, T., Dlamini, S., Mudali, P., & Kobo, H. (2019). SDN-enabled Switch Placement in Multi-domain Hybrid SD-WAN. 2019 IEEE AFRICON,

Shu, Z., Wan, J., Lin, J., Wang, S., Li, D., Rho, S., & Yang, C. (2016). Traffic engineering in software-defined networking: Measurement and management. *IEEE Access*, *4*, 3246-3256.

Sllame, A. M., & Aljafari, M. (2015). Performance Evaluation of Multimedia over IP/MPLS Networks. *International Journal of Computer Theory and Engineering*, *7*(4), 283-291.

So-In, C. (2009). A survey of network traffic monitoring and analysis tools. *Cse 576m computer system analysis project, Washington University in St. Louis*.

Srijith, K., Jacob, L., & Ananda, A. L. (2005). TCP Vegas-A: Improving the performance of TCP Vegas. *Computer communications*, *28*(4), 429-440.

Stoica, I. (2005). A Comparative Analysis of TCP Tahoe, Reno, New-Reno, SACK and Vegas. In: University of California Berkeley, CA.

Subramanian, S., & Voruganti, S. (2016). *Software-defined networking (SDN) with OpenStack*. Packt Publishing Ltd.

Tijare, P. V., & Vasudevan, D. (2016). INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY THE NORTHBOUND APIs OF SOFTWARE DEFINED NETWORKS.

Tootaghaj, D. Z., Ahmed, F., Sharma, P., & Yannakakis, M. (2020). Homa: An efficient topology and route management approach in SD-WAN overlays. IEEE INFOCOM 2020-IEEE Conference on Computer Communications,

Troia, S., Mazzara, M., Zorello, L. M. M., & Maier, G. (2021). Performance Evaluation of Overlay Networking for delay-sensitive services in SD-WAN. 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom),

Troia, S., Mazzara, M., Zorello, L. M. M., & Pattavina, A. (2021). Resiliency in SD-WAN with eBPF monitoring: municipal network and video streaming use cases. 2021 17th international conference on the design of reliable communication networks (DRCN),

Troia, S., Sapienza, F., Varé, L., & Maier, G. (2020). On deep reinforcement learning for traffic engineering in sd-wan. *IEEE Journal on Selected Areas in Communications*, *39*(7), 2198-2212.

Truong, T., Fu, Q., & Lorier, C. (2016). FlowMap: Improving network management with SDN. NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium,

Tuncer, D., Charalambides, M., Clayman, S., & Pavlou, G. (2015). Adaptive resource management and control in software defined networks. *IEEE transactions on network and service management*, *12*(1), 18-33.

Umbe, S. (2023). *VMware SD-WAN by VeloCloud: Vision, Differentiator, Future. The VmWare Networks*. The VmWare. Retrieved 10-3-2023 from https://sdwan.vmware.com/sd-wan-resources/videos/vision-differentiator-future

Venkataramani, A., Kokku, R., & Dahlin, M. (2002). TCP Nice: A mechanism for background transfers. *ACM SIGOPS Operating Systems Review*, *36*(SI), 329-343.

Versa-Network. (2023a). *Building the modern network.* The Versa Network. Retrieved 12-04-2023 from https://versa-networks.com/solutions/

Versa-Network. (2023b). *Overview of Adaptive Shaping in Versa SD-WAN deployment*. The Versa Network Group. https://academy.versa-networks.com/technical_documentat/dynamic-congestion-management-adaptive-shaping/

Victor, B. (2023). *SD-WAN - The industry standard for digital services growth.* The Metro Ethernet Forum Retrieved 12 May 2023 from https://www.mef.net/service-standards/overlay-services/sd-wan/#learn

View, M. (2023). *Silver Peak Re-Defines Scalability with NX-8000 WAN Acceleration Appliance: New Data Center Appliance Breaks down WAN Barriers to Enable Strategic IT Initiatives*. The Silver Peak. Retrieved 16-10-2006 from https://www.silver-peak.com/news/press-releases/silver-peak-re-defines-scalability-nx-8000-wan-acceleration-appliance

Wang, G., Zhao, Y., Huang, J., & Wu, Y. (2017). An effective approach to controller placement in software defined wide area networks. *IEEE transactions on network and service management*, *15*(1), 344-355.

Wang, L., Li, Q., Sinnott, R., Jiang, Y., & Wu, J. (2018). An intelligent rule management scheme for Software Defined Networking. *Computer Networks*, *144*, 77-88.

Wei, D. X., Jin, C., Low, S. H., & Hegde, S. (2006). FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Transactions on Networking*, *14*(6), 1246-1259.

Wikström, G., Persson, P., Parkvall, S., Mildh, G., Dahlman, E., Balakrishnan, B., Öhlén, P., Trojer, E., Rune, G., & Arkko, J. (2020). Ever-present intelligent communication. *Ericsson, White paper, Nov*.

Wright, G. (2023). *Scientific Method*. The Tech Target Network. https://www.techtarget.com/whatis/definition/scientific-method

Xie, J., Yu, F. R., Huang, T., Xie, R., Liu, J., Wang, C., & Liu, Y. (2018). A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges. *IEEE Communications Surveys & Tutorials*, *21*(1), 393-430.

Xu, C., Chen, S., Su, J., Yiu, S.-M., & Hui, L. C. (2016). A survey on regular expression matching for deep packet inspection: Applications, algorithms, and hardware platforms. *IEEE Communications Surveys & Tutorials*, *18*(4), 2991-3029.

Yalda, K. G., Hamad, D. J., & Ţăpuş, N. (2022). A survey on Software-defined Wide Area Network (SD-WAN) architectures. 2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA),

Yang, K., Guo, D., Zhang, B., & Zhao, B. (2019). Multi-controller placement for load balancing in SDWAN. *IEEE Access*, *7*, 167278-167289.

Yang, Z., Cui, Y., Li, B., Liu, Y., & Xu, Y. (2019). Software-defined wide area network (SD-WAN): Architecture, advances and opportunities. 2019 28th International Conference on Computer Communication and Networks (ICCCN),

Yin, X., Jindal, A., Sekar, V., & Sinopoli, B. (2015). A control-theoretic approach for dynamic adaptive video streaming over HTTP. Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication,

Yujie, R., Muqing, W., & Yiming, C. (2020). An effective controller placement algorithm based on clustering in SDN. 2020 IEEE 6th international conference on computer and communications (ICCC),

Yuniarto, R., & Sari, R. F. (2021). Performance Analysis of Multipath Deployment in Software-Defined Wide Area Network (SDWAN). 2021 International Conference on Artificial Intelligence and Computer Science Technology (ICAICST),

Zakurdaev, G., Ismail, M., & Lung, C.-H. (2022). Dynamic On-Demand Virtual Extensible LAN Tunnels via Software-Defined Wide Area Networks. 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC),

Zhang, P. (2023). *Microflow*. The Github. Retrieved 10-6-2023 from https://github.com/PanZhangg/Microflow

Zhang, Y., Tourrilhes, J., Zhang, Z.-L., & Sharma, P. (2021). Improving SD-WAN resilience: From vertical handoff to WAN-aware MPTCP. *IEEE transactions on network and service management*, *18*(1), 347-361.

Zhang, Y., Xu, C., & Muntean, G. M. (2021). A Novel Distributed Data Backup and Recovery Method for Software Defined-WAN Controllers. 2021 IEEE Global Communications Conference (GLOBECOM),

Zhu, L., Karim, M. M., Sharif, K., Li, F., Du, X., & Guizani, M. (2019). SDN controllers: Benchmarking & performance evaluation. *arXiv preprint arXiv:1902.04491*.