Impact of Integrated Machine Learning Models, Background-Traffic and Bandwidth-Limit on the Performance of Software-Defined Networking

Isiaka Babatunde SADIKU
LCU/PG/001631

Being a PhD Thesis Submitted to the Department of Computer Science, Faculty of Natural and Applied Sciences, Lead City University, Ibadan, Oyo State, Nigeria

In Partial Fulfillment of the Requirements for the Award of Doctor of Philosophy Degree (PhD) in Computer Science

2024

Certification

This is to certify that Isiaka Babatunde SADIKU with matriculation number: LCU/PG/001631 carried out this research work titled "Impact of Integrated Machine Learning Models, Background-Traffic and Bandwidth-Limit on the Performance of Software-Defined Networking" in the Department of Computer Science, Faculty of Natural and Applied Sciences, Lead City University, Ibadan, Oyo State, for the award of Doctor of Philosophy Degree (PhD) in Computer Science and that this has not been previously submitted.

_____ _____

Dr. Wilson Sakpere Date

(Supervisor)

_____ _____

Dr. Wilson Sakpere Date

(Head of Department)

Dedication

This work is dedicated to my lovely mother, father, wife and children.

Acknowledgment

I thank Allah for the sound health and understanding He granted me during my Ph.D research. I thank the Management of Lead City University for providing an enabling environment for learning and carrying out this research. I am grateful to my supervisor Dr. Wilson Sakpere, the Head of the Department, for his valuable intellectual suggestions that greatly aided my research. I am also thankful to the Postgraduate Coordinator, Dr. A. Waheed, for standing by me during this research. I acknowledge the staff members from the Department of Computer Science, Lead City University in Ibadan for their academic and moral support during this research: Dr. V. B. Oyekunle, Dr. A. M. Ayoade, Dr. A. F. Afe, Dr. A. T. John- Dewole, I. M. Iyanu, Dr. W. Ajayi, Dr. R. A. Badru, and Prof. S. O. Akinola. Your constructive criticism has greatly contributed to the success of this work. I am grateful to postgraduate colleagues in the department: Dr. Kolapo, Dr. Fikayo Awodele, Dr. Nonny-N, Dr. Olufemi, Oniovosa, Felicia Adelodun, Kofoworola Ayeni, and Dr. John-Dewole, for their helpful discussions and advice. I thank Prof. A. O. Oluwatope of OAU and Dr. L. Olawoyin of Unilorin for their advice on the use of a simulator for this research. I express my gratitude to E. Idowu-Agida, E. O. Solaru, Dr. A. Musari, Prof G. B. Onolaja, Prof Achimugwu, Amir Salim, GMC members, and all the staff in the Department of Computer Science at Gateway Polytechnic, Saapade: T. K. Ogunyinka, E. Biya, E. Idowu-Agida, O. O. Oni, O. O. Oyekunle, O. O. Onalaja, O. A. Aina, Z. A. Mahmood, S. P. Olorunda Seyi, Olasunkonmi A. Ogunniyi, Gbemi Daniel and Dr. A. M. Logunleko - thank you all for your assistance. I am grateful to my dearest parents, my late father, Seriki Adini of Ligbein Ode-Remo, and my mother, Ajisafe Adini of Ligbein for their support every time. I thank my siblings; Fumilola, Fausat, Akeem, Musliu, Rafiu, and Jelil for being there for me always. Also, special thanks to my in-laws especially Basirat, Opeyemi, and Teslim for their support every time. I am grateful to my wife, Dr. N. A. Sadiku, Head of Department, Forest Resource Management at Unilorin, for her steadfast support. Special thank to my wonderful and lovely children: M. Zainudeen, H. Olalekan, and I. Abdul-Quadri. Though the above-mentioned institutions and persons have assisted in the process of this research work, I alone stand responsible for the errors, if any, found in the work.

Abstract

Efficient data flow in computer networks is crucial for modern applications, but network performance faces challenges due to the complexity of network types and configurations. Understanding the impact of different networking approaches on packet flow, bandwidth, latency, jitter, and throughput is essential for improving network performance. Traditional Computer Networks (TCN) and emerging technologies like Software-Defined Networking (SDN) have distinct advantages and trade-offs in terms of bandwidth usage, latency, throughput, and jitter. This study aims to assess the influence of background traffic, bandwidth limits, and dataflow features on SDN performance and the ability of machine learning models to predict network behavior. The analysis reveals several key findings: Traditional networks exhibited higher throughput, while hybrid TCN-SDN showed reduced bandwidth usage. Latency varied across network types, with SDN networks showing potential increases. Jitter was significantly impacted by non-homogeneous networks, raising concerns about overall performance stability. ANOVA and Duncan's tests confirmed the importance of latency, bandwidth, and throughput in influencing network behavior. Back-ground traffic and bandwidth limits were shown to have a complex relationship with SDN performance, particularly in terms of TCP bandwidth, throughput, and latency. Correlation analyses highlighted strong relationships between network parameters, providing deeper insights into dataflow dynamics. Among machine learning models, Support Vector Machine with Radial Basis Function Kernel (SVM_RBF) consistently outperformed others, while the stacked 5-stacked model demonstrated superior accuracy in predicting SDN performance across different datasets and scenarios. This study offers valuable insights into the interplay of network types, traffic conditions, and performance metrics. The results indicate that while traditional networks offer higher throughput, hybrid TCN-SDN configurations present advantages in bandwidth efficiency but may incur higher latency. The machine learning models successfully predicted network performance, with the 5-stacked model emerging as the most accurate across a range of conditions.

Keywords: Performance Metrics, Programmable Network, Data Flow, Machine Learning, Bandwidth-traffic

Word Count: 290 words

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviations	Meaning
AC	Access Control
AP	Access Point
BN	Number of Bytes
CC	Cloud Computing
CDNs	Content Delivery Networks
DNS	Domain Name System
DT	Decision Tree
EL	Ensemble Learning
FD	Flow Destination
FS	Flow Source
FTP	File Transfer Protocol
HML	Hybrid Machine Learning
IA	Idle Age
IaaS	Infrastructure as a Service
IoT	Internet of Thing
IP	Internet Protocol
KNN	K-Nearest Neighbour
LAN	Local Area Network
LB	Load Balancing
MAN	Metropolitan Area Network
MCD	AMulti-Criteria Decision Analysis
ML	Machine Learning
MODEM	Modulator-demodulator
MPL	SMultiprotocol Label Switching
NAT	Network Address Translation
NFV	Network Functions Virtualization
NOS	Network Operating System
OF	OpenFlow switch
PaaS	Platform as a Service
PAN	Personal Area Network
PC	Personal Computer
PI	Input Port
PN	Number of Packets
QoS	Quality of Service
RF	Random Forest
RL	Reinforcement Learning
RTT	Round-trip Time
SaaS	Software as a Service
SD-WAN	Software-defined Wide-area Network
SDN	Software-defined Networking
SOM	Self-Organizing Map
SRAFM	Service and Resource Aware Flow Management
SVM	Support Vector Machine
SW	Switche
T	Flow Time
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

VM	Virtual Machine
VoIP	Voice-over Internet Protocol
VPN	Virtual Private Network
WAN	Wide Area Network

Chapter One
Introduction

1.1 Background to the Study

Computer networks and the Internet are revolutionizing how people communicate, learn, work, and play. The demand for unlimited communication services continues to increase for various man needs for research, education, businesses, online banking, online gaming, shopping, and other media networking applications[1,2,3]. Thus, the Internet network architecture would become too complicated and gigantic if the communication network is not competent or intelligent enough to remove complexity in the computer networks[4]. Intelligent systems or innovative software enables control and automate the operation of different gadgets through which the user can manage and perform the required function at the immediate or remote point[5,6,7]. Expanding the existing network is relatively expensive, while the demand for network-based services is increasing rapidly[8,9]. The application of machine learning plays a vital role in decision-making in operating computer networks.

Software-Defined Networking (SDN) is becoming popular, especially in high-bandwidth and dynamic applications like cloud computing[10]. Software-Defined Networking (SDN) technology enables continuous network operation, development, and deployment. SDN has many advantages, such as centralized monitoring that helps reduce manual communication with the hardware for enhanced network efficiency. Additionally, separating the control plane and data plane leads to simpler hardware management, which increases the chances of having more expertise among hardware vendors, as the SDN does not rely solely on commercial software. SDN decouples the control and data planes from the network core devices for operating purposes[11]. The control plane (i.e., SDN controller) acts as a Network Operating System (NOS[12]. The data plane resides inside the network core devices and is only responsible for forwarding data packets controlled by the central controller. This segregation increases the agility and flexibility of a networking

infrastructure and reduces its operational expenses[13].

SDN consists of the applications plane, controllers plane, and devices or infrastructure plane. Applications planes relay information about the network or requests for specific resource availability or allocation. The application layer embodies the implementation of typical workings of the network or functions. The application layer can appropriately be adapted to the changing business requirements. The SDN controller communicates with applications to determine the destination of data packets[14]. The networking devices plane consists of various network devices, both physical and virtual. The device plane receives and forwards instructions from the controllers regarding how to route the packets. In traditional networks, the control and data plane are coupled in the same device. The primary role of these network devices plane in SDN is to forward the data, providing a very efficient and flexible forwarding mechanism[15]. In an SDN, switches receive routing and traffic management decisions via a control channel from a centralized controller[16]. Traditional computer networks do not allow a fast evolution towards a process that contributes to improving the online transaction of services. Traditional Computer Networking consists of computers primarily implemented from dedicated devices using one or more switches, routers, and application delivery controllers[17].

SDN is more flexible, allowing users greater control and ease of managing resources virtually through the control plane. In contrast, traditional networks use switches, routers, and other physical infrastructure to create connections and run the network[18]. SDN controller communicates with applications like firewalls or load balancers via its northbound APIs. The controller talks with individual network devices using a southbound interface to configure network devices and choose the optimal network path for application traffic[19,20]. As a result of this communication, application developers program the network directly instead of using traditional protocols[21]. SDN uses an open, flexible, and dynamic architecture defined through different software programming languages[22].

In traditional legacy network devices, the control decisions unit and the forwarding unit are tightly coupled, where both the control decisions, like optimal route calculation and forwarding, occur in the same device. The hardware of these devices is made specifically for a particular task. They are not flexible enough to allow researchers to test new algorithms that they might come up with to solve any of the networking issues. This requires researchers to create custom hardware and have a new setup for each experiment. It would be much better if commercial switch providers allowed more flexibility, thus allowing researchers to test their new ideas on the same network without new hardware. OpenFlow switch (OF) enables this flexibility. SDN de-couples the control and forwarding unit, i.e., the data plane. This is useful for many purposes like data centers as it reduces the cost of maintenance as well as for researchers to be able to carry out various network experiments. The center has a compassionate nature of operation workloads. As a result, data center operators often desire to quickly identify bottleneck network links and route data traffic around them at very fast timescales to serve the network communities at high network capacity without cost overburdening the network provider and the clients. For this to occur smoothly, network operators must be willing to embrace SDN to tackle the lack of precise and robust control, access, visibility, and flexibility often experienced in TCN.

OpenFlow is the protocol used to manage the switch to be able to add, remove, modify flow entries, capture flow statistics, etc[23]. SDN controller is the user program that uses OpenFlow to communicate with the OpenFlow switch. There are various frameworks available for writing controller applications. Examples are Ryu, POX, OpenDaylight, etc[24]. OpenVSwitch is an open virtual switch that supports OpenFlow switches and other switch management protocols. In an OpenFlow switch, forwarding decisions can be taken based on MAC, IP, in-port, VLAN_ID, etc. This makes the OpenFlow switch act as a catalyst switch that can operate at layer 2 or layer 3 of the OSI model[25]. OpenFlow controllers like NOX, POX, Beacon, Floodlight, Ryu Open Daylight, etc. POX is very popular for prototyping and its simple structures[26,27]. POX is an OpenFlow-based

controller derived and developed through Python programming language that aims to provide an efficient and accessible environment for performing research investigations and tests in SDN networks. POX relies on the component-based model in which the whole network elements, as well as activities are recognized as separate components that can be isolated and utilized every time the need arises. The location of POX is specifically between network components on one side and the applications on the other. POX is responsible for achieving any type of communication between applications and devices. SDN controllers mainly rely on protocols such as OpenFlow protocol, where network devices are configured, optimal network paths for traffic applications are selected, and servers are permitted to inform the switches where to direct the packets. In addition to being a framework for the interaction with the OpenFlow switches, POX can be utilized as the foundation for some of the ongoing work to assist in building the emerging discipline of SDN, where it can be applied in various fields such as distribution prototyping and exploration, SDN debugging, network utilization, controller design, and programming models. POX modules are, in fact, extra Python programs that can be used in case POX is initiated from the command line prompt; these modules carry out network functionality in SDN[28].

Simulation and emulation network platforms play an important role in studying and evaluating different networks design and performance[29]. Mininet is a network emulator that can be used to create any topology with multiple hosts and switches[30,31]. Mininet is the most popular SDN platform in the form of a virtual test used for testing network tools and protocols[32,33]. Quality of Service (QoS) is a set of technologies on a computer network that guarantees its ability to run high-priority applications and traffic reliably under limited network capacity[34]. QoS technologies provide differentiated handling and capacity allocation to specific flows in network traffic to assign handling of packets and afford bandwidth to that application or traffic flow[35]. Performance parameters commonly measured in QoS are bandwidth (throughput), latency (delay), jitter (variance in latency), and error rate. Internet Protocol (IP) packet (including pings) size ranges from 1 to

65,507 bytes. Ping is the most common network administration utility used to test the reachability of a host on an IP network and to measure the Round-trip Time (RTT) for messages sent from the originating host to a destination computer. Ping sends ICMP echo requests/replies to test the connectivity to other hosts. Standard ICMP ping shows that the computer host is responding or otherwise unreachable.

Computer network performance monitoring is essential for smoothly running global computer network communities. Iperf is a tool for IP network performance. It is a cross-platform tool that runs on major operating systems. Iperf supports the measurement and tuning of various parameters related to timing, buffers, and network protocols like TCP, UDP, and SCTP with IPv4 and IPv6. Iperf has client and server functionality and can create data streams to measure throughput between hosts in one or both directions.

Background-Traffic refers to traffic flows that generally run for a long time and are not sensitive to latency, for example, the video streaming or running torrent service to which malicious traffic attempts to hide or run along the traffic flow. Normal or legitimate traffic usually refers to forms of desired internetwork traffic. Some of these regular traffic are primarily short-lived and don't have stringent requirements on packet drop and latency. Performance of the flow Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) has been evaluated using throughput as a metric[36].

Bandwidth is the amount of data that can be transferred from one source to another source in a computer system or computer network. For example, a website that has high traffic needs lots of bandwidth to allow smooth transfer of data to and or from the requesting machine. Bandwidth can be described in terms of up-link and down-link. Performance analysis of the network is done by analyzing bandwidth utilization between the hosts. An analysis of TCP and UDP bandwidth between hosts is achieved by executing internet performance tests like Iperf. Iperf is a network testing tool that can create TCP and UDP data streams and measure the throughput of a network

carrying them. Iperf measures TCP bandwidth and allows the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay jitter, and datagram loss between hosts[37]. A throughput is affected by TCP. Wireless signal loss adjustment due to contention and communicating the random loss to make the congestion window insensitive enhance the traditional TCP throughput and increased bandwidth utilization[38]. Unlike TCP, UDP throughput is affected by different network parameters - out-of-order delivery of packets, network jitter, packet loss during transmission, etc. Network jitter is the deviation in time for periodic arrival of data-grams. Reducing unwanted network load can minimize packet loss during communication [39,40].

Machine Learning (ML) has been used to manage and support various business operations[41]. ML can be used to predict traffic flows, generate more brilliant flow analysis, monitor network health, tighten security measures, etc[42]. The models that are used for the prediction are developed from ML algorithms that learn from the data set. ML models could be weak or strong learners. Combining weak learners or weak learners with strong learners makes more accurate predictions than any single learner alone or fast learning with better prediction. Hybrid Machine Learning (HML) is an improved machine learning that combines algorithms, methods, processes, or procedures from similar or different application areas to complement each other and produce more accurate processes[43]. HML complements candidate methods and uses one to overcome the weakness of the others[44]. The combined algorithms use any averaging (simple or weighted) methods and voting (majority or weighted) for regression and classification methods[45]. Machine Learning (ML) algorithms have several opportunities in SDN and have been used for flow detection and classification, security, and traffic management using different machine learning algorithms[46,47]. This study presents the performance of combined machine learning on the influence of Background-Traffic and Bandwidth-Limit on Software-Defined Networking.

Ensemble learning techniques are used in some applications to enhance the performance of single-

classifier systems[48,49]. Ensemble learning is the process that allows multiple models, such as classifiers or experts, to be strategically generated and combined to solve a particular computational intelligence problem, primarily to improve the performance of a model or reduce the likelihood of an unfortunate selection of a poor one like classification, prediction, function approximation, data fusion, incremental learning, non-stationary learning, and error-correcting[50]. Ensemble learning first obtains a set of features with a variety of transformations[51]. Based on these learned features, multiple learning algorithms produce weak predictive results[52]. Finally, ensemble learning fuses the informative knowledge obtained from the above results to achieve knowledge discovery and better predictive performance via voting schemes in an adaptive way. The domains of these applications include classification and regression problems. Random forest models and gradient boosting models are well-known ensemble models; they use a combination of weak learners to build up an ensemble[53]. In these models, the collection of weak learners is homogeneous; the same types of weak learners are grouped together to show their combined strength[54,55]. Bagging and boosting are two popular categories of ensemble learning[56]. Random forest is the popular ensemble learning model in the bagging category. AdaBoost is another popular ensemble learning model that comes under the boosting category.

1.2 Statement of the Problem

As computer networks continue to expand rapidly and applications grow increasingly complex, the demand for network performance has intensified. Managing and optimizing network performance presents a significant challenge, particularly in understanding the impact of background traffic and bandwidth limitations. Background traffic, generated by routine or non-critical processes, can significantly affect the performance of critical applications, especially when faced with bandwidth constraints. Conventional methods for analyzing and managing network performance often rely on traditional statistical models and manual interventions, which may not comprehensively capture the

complex and dynamic nature of network traffic. Therefore, there is urgent need for more advanced, adaptive, and automated approaches to accurately predict and mitigate the negative impacts of background traffic and bandwidth constraints.

1.3 Aim and Objectives of the Study

This research aims to assess the impact of background traffic with bandwidth limits on the performance of Software-Defined Networking (SDN) and the implications of integrating machine learning models into Software-Defined Networking.

The specific objectives of this research are to:

i. examine the effect of Traditional Computer Networking (TCN), Software-Defined Networking (SDN), and hybrid TCN-SDN on performance of computer network, focusing on packet flow and application running on computer network

ii. evaluate the influence of background traffic and bandwidth limitations on the performance of Software-Defined Networking (SDN), specifically examining metrics such as latency, bandwidth, throughput, jitter, and datagram loss.

iii. develop and evaluate the performance of stack machine learning models amid the influence of background-traffic and bandwidth-limit on Software-Defined Networking (SDN).

1.4 Research Question

For this research, the following research questions were raised;

i. How should the levels of bandwidth limitations for applications running on a computer network be determined in a situation where the network performance (latency, bandwidth, throughput, and jitter) starts to degrade noticeably for Traditional Computer Networking (TCN) and Software-Defined Networking (SDN)?

ii. Why is the influence of background traffic and bandwidth limitations on the performance of

Software-Defined Networking (SDN) significant?

iii. How could machine learning models (KNN, SVM_RBF, DT, RF, MLP, and stack models) accurately predict network performance under varying conditions of background traffic and bandwidth limitations for optimized network performance in real time?

1.5 Significance of the Study

Generally, computer networking promotes various activities of human beings by allowing them to communicate and share information effortlessly, increasing productivity, efficiency, flexibility, and data security.

This study provide information on the limitations of traditional computer networks. The limitation is becoming enormous as the number of users of computer networks is increasing day by day with the growth of data communication. For example, as the number of network users grows, the devices added to the network also increase and become huge, requiring an update in the network configuration. This vast network update, which is error-prone, with inconsistent network policies, security, and inability to scale, is manually configured by network administrators and integrated into the production network. It has been observed that traditional network architecture is gradually unable to meet today's demand for network business, and the existing problems are becoming increasingly prominent[57]. The current traditional network is becoming complex due to the inclusion of many devices produced by different manufacturers, which usually require different ways to debug, configure, manage, and deploy as a production network. The traditional network cannot perform intelligent flow control and visualize network status.

Software-Defined Networking (SDN) is a decoupled architecture layers in a communication network that makes the computer network more programmable, which improves the level of network resource pooling while realizing automatic network deployment, configuration, support

rapid business launch, and flexible expansion of the computer network. The introduction of programmable features helps to achieve automated network services and protocol scheduling by bridging the LAN (either in virtual or physical modes) to SDN. The SDN, which is a programmable computer network, is handy for network researchers to carry out various computer network experiments. Performance of SDN under Bandwidth-Limit and some unaccounted processes utilizing network resources were observed and analyzed, and conclusions were presented in this study. Information obtained on the effect of Background-Traffic and Bandwidth-Limitation on SDN will assist network administrators and other stakeholders in computer networks in deploying, monitoring, and updating SDN for high performance.

1.6 Scope of the Study

The study uses personal computers (s) and 'Mininet' as emulation systems to implement networking of virtual PCs with physical PCs to the Internet. Mininet was installed as an emulation system on a PC known as 'Mininet PC' that was used to set up a connection of virtual PC using Python code. A software-defined network was set up on the 'eminent PC' consisting of a controller and open switches with PCs on each of the switches. In the Python code, the openvswitches were looped to form a stacked switch. One edge switch was bridged with the first access point that connected the physical PCs, while the other edge switch was bridged with another access point that connected to the Internet. After the connectivity test, the network experiments were set up for Background-Traffic and bandwidth tests to measure Qos (throughput, bandwidth, latency, and jitter) for computer network performance. The observed results were used as dataset. The datasets obtained were cleaned, trained, and tested in a jupyter notebook environment for stack machine learning.

1.7 Limitations of the Study

Due to some of the challenges in the SDN architecture, this research was based on the use of wireless LANs integrated with SDN implemented using an emulation system (Mininet). The Mininet was installed on a personal computer running Ubuntu 22.04 LTS to implement SDN consisting of networking of virtual end devices like PCs, openflow switches, and SDN controller. Python programming was used to implement network typologies. The switches were stacked to achieve enough network packet flow. The network traffic was monitored using the ping command and Wireshark and, after that, analyzed.

1.8 Operational Definition of Terms

While all terms are defined as they are used in the thesis, this section highlights the most prevalent terms and explains their meaning as used in this thesis. The most predominant terms in this study include background network traffic, Bandwidth-Limit, emulation software, virtual machine, and Mininet: software-defined wide-area network, data packet, and SDN network topology

i. Emulation Software

Emulation software enables running an operating system on a hardware platform for which it was not engineered initially or running application programs on different operating systems other than those for which they were initially written. Emulators are hardware or software platforms (also known as hosts) that allow a computer system to behave like another machine running applications and services designed for the latter (also known as guest)[58]. There are many commercial and open-source emulators available for major operating systems. For instance, WINE is a tool that enables Windows applications to be run on Linux and Mac systems. Dolphin is an application that allows Nintendo GameCube and Wii games to be played on a computer. BlueStacks is an emulator that makes it possible to use Android apps on Windows and Mac. Xcode emulator allows users to run iOS on Mac and Windows. Appetize.io is a browser-based emulator that will enable iOS apps on

any PC

ii. Mininet

Mininet is an emulation software that can be used to create a realistic virtual network, running a real kernel, a network of virtual hosts, switches, controllers, links, and application code[59]. Presently, Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking. Mininet supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network on a personal computer like a laptop. In this study, Mininet was run on a single machine as native, though it can also be run on a virtual machine (VM) or in the cloud[60]. Using Mininet is one of the ways to develop, share, and experiment with Software-Defined Networking (SDN) systems using OpenFlow. Mininet is actively developed, supported, and released under a permissive BSD Open Source license.

iii. Virtual Machine

A Virtual Machine (VM) behaves like a computer, running as a separate computing environment capable of running a different operating system while relying on the host computer architecture to function as a new computer[61]. Virtual machines can be process VMs and system VMs. Process virtual machine allows a single process to run as an application on a host machine, providing a platform-independent programming environment by masking the information of the underlying hardware or operating system. An example of a process VM is the Java Virtual Machine, which enables any operating system to run Java applications as if native to that system. A system virtual machine is a virtualization type that substitutes for a physical machine. A system platform supports the sharing of a host computer's physical resources between multiple virtual machines, each running its copy of the operating system. This virtualization relies on a hypervisor, which can run on bare hardware (such as VMware ESXi) or on top of an operating system (such as Virtualbox).

Components of IT infrastructure are visualized nowadays using various specific types of virtualization like hardware virtualization, software virtualization, storage virtualization, network virtualization, and desktop virtualization.

iv. Bandwidth

Bandwidth, data, and speed are often used interchangeably and are closely related terms. Bandwidth refers to the amount of data allocated or pulled per second. The bandwidth a computer connection is allotted determines how much data can be downloaded per second or transferred from the Internet to a computer. A Bandwidth-Limit refers to a speed limit or data limit in a network. For instance, broadband Internet service providers sell access plans based on speed, limiting bandwidth according to each plan. Plans with higher limits are more expensive but transfer faster, though. There is unlimited access by some telcos, and users don't have to worry about limits in terms of data download, yet there is still a Bandwidth-Limit. Apart from internet usage, Bandwidth-Limits might also apply to personal domains. SDN features, via its flexible and programmable characteristic, bandwidth usage for connecting computing devices[62].

v. Background Network Traffic

Background network traffic (or Background-Traffic) is one of the terms in network security controllers testing where it refers to the non-malicious (that is legitimate) traffic that was passing in parallel to the malicious (that is illegitimate) traffic[63]. The logic of using background network tests is to access network performance in a real-life environment when normal and potentially malicious traffic flow together. Network traffic also refers to data traffic, the amount of data moving across a computer network at any given time.

vi. Network Traffic

There are two directional flows (i.e., north-south flow and east-west flow) in network traffic. North-south traffic refers to client-to-server traffic that moves between the data center and the rest of the

network (i.e., a location outside the data center). East-west traffic refers to traffic within a data center, also known as server-to-server traffic[64]. Traffic affects network quality due to unusually high traffic that can lead to slow download speeds or spotty voice-over internet protocol (VoIP). Network traffic is categorized as real-time and non-real-time[65]. Real-time traffic is essential or critical to business operations and must be delivered on time and with the highest quality possible. Examples of traffic include VoIP, videoconferencing, and web browsing. Non-real-time traffic is also known as best-effort traffic, which network administrators consider less critical than real-time traffic. For example, traffic includes File Transfer Protocol (IP) for web publishing and email applications.

vii. Data Packets

When data travels over a network or the internet, it is broken down into smaller batches so that larger files can be transmitted efficiently. Information (raw data) passing over a network or over the internet is broken down, organized, and bundled into data packets into smaller batches so that larger files can be transmitted efficiently and reliably through the network and then opened and read by another user in the network as detailed in the OSI layers protocols. Each packet takes the best route possible to spread network traffic evenly. In order to better manage bandwidth, network administrators usually prioritize traffic flow to be treated by network devices like routers, switches, and SDN controllers[66].

viii. Software-defined Wide Area Network

A Software-defined Wide-area Network (SD-WAN) uses programmable software to manage network communication between an organization's data centers and remote locations[67]. As a programmable network, SD-WAN can be used to automatically segment traffic based on defined criteria and accommodate multiple connection types such as Multiprotocol Label Switching

(MPLS) and Long Term Evolution (LTE)[68].

ix. SDN Network Topology

Network topology refers to the physical or logical arrangement of nodes and connections in a network, often represented as a graph. Administrators use network topology diagrams to determine the best placements for each node and the optimal path for traffic flow. Hierarchical computer network topology assists network administrators in quickly locating faults and fixing issues, thereby improving data transfer efficiency. Network topology plays a significant role in how a network functions. Several types of topologies exist; for instance, physical topologies include bus topology, where every node is connected in series along a single cable; star topology, where a central device connects to all other nodes through a central hub; ring topology, where the nodes are connected in a closed-loop configuration while the loop (i. e. rings) can pass data in one direction only or are capable of transmission in both directions. The bus, start, and ring topology can sometimes be meshed. Mesh network topology links nodes with connections so that multiple paths between at least some network points are available[69]. A network can be fully meshed when all nodes are directly connected to all other nodes and partially meshed when some nodes have multiple connections to others. Meshing multiple paths increases resiliency but also increases cost. Bus, star, and ring topology are also used to form linear and star topologies. Star topologies are constructed to form a tree network topology consisting of one root node, and all other nodes are connected in a hierarchy. Ethernet switch networks, particularly data center networks, are commonly configured in a tree-like structure. This allows for a more extensive and organized network, ensuring efficient data transmission. Simple linear topology and fat tree topology consisting of switches and hosts have been used in SDN topology to reduce controller load while delivering identical discovery functionality[70,71].

x. Cloud Computing

Cloud computing is a transformative paradigm in Information Technology (IT) that involves the delivery of various services, including storage, computing power, and applications, over the internet. This model enables users to access and utilize resources on a pay-as-you-go basis, eliminating the need for physical infrastructure and reducing the complexities associated with traditional IT setups. Cloud computing is characterized by its scalability, flexibility, and cost-effectiveness, allowing businesses and individuals to scale their operations seamlessly, adapt to changing demands, and only pay for the resources they consume[72]. The cloud encompasses a range of deployment models, including public, private, and hybrid clouds, each catering to specific needs and preferences[73]. Public cloud services are offered by third-party providers on a shared infrastructure, private clouds are dedicated to a single organization, and hybrid clouds combine elements of both[74]. The cloud computing ecosystem includes Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), offering a spectrum of services to meet diverse requirements. Despite its numerous benefits, such as increased efficiency and accessibility, cloud computing also raises concerns about data security, privacy, and regulatory compliance[75]. As technology continues to advance, cloud computing remains a critical driver of innovation, revolutionizing the way businesses operate and individuals access and interact with digital resources.

xi. Traditional Computer Networks

Traditional Computer Networks (TCN) and Software-Defined Networking (SDN) are two approaches to network architecture that have distinct characteristics and functionalities.

Traditional computer networks rely on a centralized management model where network devices, such as routers and switches, control and direct the flow of data packets. These networks typically require manual configuration and management of individual network devices[76]. The infrastructure is built using physical hardware components, and the network behavior is determined by the

firmware and software running on these devices. Traditional networks follow a hierarchical or distributed architecture, and changes to the network configuration usually involve making changes to each individual device.

xii. Software-Defined Networking

Software-Defined Networking (SDN) is a more flexible and programmable approach to network management[77]. SDN separates the control plane from the data plane, allowing the network control logic to be centralized and programmatically controlled. In an SDN architecture, a centralized controller is responsible for managing the network and making decisions about how data packets should be forwarded. The controller communicates with network devices through an open and standardized protocol, such as OpenFlow, to configure and control their behavior. This centralized control allows for easier network management, dynamic provisioning, and more efficient use of network resources. SDN provides several benefits over traditional networks. It enables network administrators to have a global view of the network and make changes easily and quickly[78]. Network policies can be implemented and enforced centrally, improving security and reducing configuration errors. SDN also allows for network virtualization, where multiple virtual networks can coexist on a shared physical infrastructure, enhancing resource utilization. While traditional computer networks are still widely used and offer stability and reliability, SDN is gaining popularity due to its flexibility, scalability, and programmability. It provides a foundation for network automation, orchestration, and the implementation of innovative network services and applications. SDN is often seen as a key enabler for modern network technologies like Software-Defined WAN (SD-WAN) and Network Function Virtualization (NFV)[79].

Endnote

1. S. Makridakis, "The Forthcoming Artificial Intelligence (AI) Revolution: Its Impact on Society and Firms" Futures, Elsevier, 2017, 90, 46-60

2. M. Meeker and L. Wu "Internet Trends" Kleiner Perkins Menlo Park, CA, USA, 2018

3. J. Firth, J. Torous, B. Stubbs, J. A. Firth, G. Z. Steiner, L. Smith, M. Alvarez-Jimenez, J. Gleeson, D. Vancampfort & C. J. Armitage “The Online Brain: How the Internet may be Changing our Cognition” World Psychiatry, Wiley Online Library, 2019, 18, 119-129

4. C. Guevara and M. Santos "Intelligent Models for Movement Detection and Physical Evolution of Patients with Hip Surgery", Logic Journal of the IGPL 29, 6, 2021, pp. 874--888.

5. A. S. Wumi and A. Oludele, “Evolution of Digital Edifices: from Shanks and Adobe to Smart and Intelligent Edifice; a Trail to the Future” International Journal of Multidisciplinary Sciences and Engineering, 2017, 8, 11 - 17

6. W. Sakpere, M. Adeyeye-Oshin and N. B. W Mlitwa "A state-of-the-art Survey of Indoor Positioning and Navigation Systems and Technologies", South African Computer Journal 29, 3, 2017, pp. 145 -197

7. Y. Zhang, J. Liu and W. Shen "A Review of Ensemble Learning Algorithms Used in Remote Sensing Applications", Applied Sciences 12, 17, 2022, pp. 8654.

8. B. R. Dawadi, A. Thapa, R. Guragain, D. Karki, S. P. Upadhaya and S. R. Joshi "Routing Performance Evaluation of a Multi-domain Hybrid SDN for its Implementation in Carrier Grade ISP Networks", Applied System Innovation 4, 3, 2021, pp. 46

9. M. Feurer and F. Hutter "Hyperparameter Optimization, in Automated Machine Learning” Springer, Cham, 2019, pp. 3--33.

10. Y. Xiao, M. Varvello and A. Kuzmanovic. Monetizing Spare Bandwidth: The Case of Distributed VPNs. Proceedings of the ACM on Measurement and Analysis of Computing Systems, ACM New York, NY, USA, 2022, 6, 1-27

11. X. Zhao, M. Jahre, Y. Tang, G. Zhang and L. Eeckhout "NUBA: Non-Uniform Bandwidth GPUs", Association for Computing Machinery, 2023, 544-559.

12. A. K. Arahunashi, S. Neethu and H. V. R. Aradhya. Performance Analysis of Various SDN Controllers in Mininet Emulator IEEE Xplore, 2019

13. L. Nacshon, R. Puzis and P. Zilberman "Floware: Balanced Flow Monitoring in Software-defined networks", arXiv preprint arXiv:1608.03307, 2016

14. S. Jitnukulsiri and C. Aswakul. Validation of SDN Emulator Based on Mininet and ONOS Controller for IEC 61850 Packet Delay Measurement. IEEE International Conference on Consumer Electronics -Asia (ICCE-Asia), 2019

15. K. Raghunath and P. Krishnan "Towards a secure SDN architecture", International Conference on Computing, Communication and Networking Technologies (ICCCNT) 2018, pp. 1-7.

16. H. Nugroho, M. Irfan and A. Faruq. Software-defined Networks: a Comparative Study and Quality of Services Evaluation. Scientific Journal of Informatics, 6, 182, 2019

17. D. S. Rana, S. A. Dhondiyal and S. K. Chamoli "Software-Defined Networking (SDN) Challenges, Issues and Solution", International Journal of Computer Sciences and Engineering 7, 1, pp. 884–889, 2019

18. K. Y. Alghamdi. A Software-defined Paradigm for Mobile Networks: A Feasible SDN Based Architecture Solution for 5G Networks, Doctoral Dissertation, 2021

19. E. Ahvar, S. Ahvar, S. M. Raza, S. V. J. Manuel & G. M. Lee “Next generation of SDN in cloud-fog for 5G and beyond-enabled applications: Opportunities and challenges” Network, MDPI, 1, 28-49, 2021

20. L. Nkenyereye, Q. Pham and J. Song "Efficient RSU Selection Scheme for Fog-based Vehicular Software-Defined Network", IEEE Transactions on Vehicular Technology, 2021

21. K. Poularakis, L. Tassiulas and T. V. Lakshman "Modeling and Optimization in Software-Defined Networks", Synthesis Lectures on Learning, Networks, and Algorithms 2, 2 , pp. 1-174, 2021

22. K. Y. Alghamdi. A Software-defined Paradigm for Mobile Networks: A Feasible SDN Based Architecture Solution for 5G Networks, Doctoral Dissertation, 2021

23. J. Miguel-Alonso "A Research Review of OpenFlow for Datacenter Networking", IEEE Access 2022

24. Y. Li, X. Guo, X. Pang, B. Peng, X. Li and P. Zhang. Performance Analysis of Floodlight and Ryu SDN Controllers under Mininet Simulator. IEEE/CIC International Conference on Communications in China, 2020

25. V. K. Rathi and K. Singh. SDN Layer 2 Switch Simulation Using Mininet and OpenDayLight. Springer Nature Singapore, 2018, 319-327

26. C. Fancy and M. Pushpalatha. Performance evaluation of SDN controllers POX and floodlight in mininet emulation environment. IEEE Xplore, Proceedings of the International Conference on Intelligent Sustainable Systems (ICISS 2017), 2017

27. K. K. Sharma and M. Sood. Mininet as a Container Based Emulator for Software Defined Networks. International Journal of Advanced Research in Computer Science and Software Engineering, 2014

28. R. Jawaharan, P. M. Mohan, T. Das and M. Gurusamy. Empirical Evaluation of SDN Controllers Using Mininet/Wireshark and Comparison with Cbench. IEEE 978-1-5386-5156-8/18, 2018

29. T. N. N. Khanh & D. V. Khoa Emulation of software-defined network using Mininet. Dalat University Journal of Science, 2021, 80 -92

30. J. Bhatia and P. Patel. Mininet--An Emulator for Prototyping Large Network Topologies on a Single Machine. Admin Insight, 2015, pp. 51--56

31. R. Barrett, A. Facey, W. Nxumalo, J. Rogers, P. Vatcher and M. St-Hilaire. Dynamic Traffic Diversion in SDN: testbed vs Mininet. In 2017 International Conference on Computing, Networking and Communications (ICNC) (pp. 167-171). IEEE., 2017

32. E. Sturzinger and S. Cilenti. A Hybrid Software-defined Network Platform for Undergraduate Research and Education. Proceedings of The National Conference On Undergraduate Research (NCUR) Kennesaw State University, Kennesaw, Georgia , April 11-13, 2019

33. M. Jadin, O. Tilmans, M. Mawait and O. Bonaventure. Educational Virtual Routing Labs with IP Mininet. in ACM SIGCOMM Education Workshop, 2020

34. X. Yuan, Z. Liu, Y. Park, H. Hu and H. Li. Teaching SDN Security Using Hands-on Labs in

CloudLab. Journal of The Colloquium for Information Systems Security Education vol. 7, no. 1, 2020, pp. 6 - 6.

35. G. Nikandish, R. B. Staszewski and A. Zhu . Breaking The Bandwidth Limit: A Review of Broadband Doherty Power Amplifier Design for 5G. IEEE Microwave Magazine 21, 4, 2020, pp. 57--75.

36. M. H. H. Khairi, S. H. S. Ariffin, N. M. A. Latiff and K. M. Yusof . Generation And Collection Of Data For Normal And Conflicting Flows In Software-defined Network Flow Table. Indonesian J. Electr. Eng. Comput. Sci, 22(1), 307, 2021

37. B. Zieliński . Assessment Of IPerf As A Tool For LAN Throughput Prediction", International Journal of Electronics and Telecommunications, 2023, pp. 523--528

38. J. R. S. Veluswami . Improving Throughput of Transmission Control Protocol Using Cross Layer Approach. Computer Systems Science & Engineering, 43(3), 2022

39. I. Z. Bholebawa, R. K. Jha and U. D. Dalal . Performance Analysis Of Proposed OpenFlow-based Network Architecture Using Mininet. Wireless Personal Communications, Springer, 2016, 86, 943 - 958

40. O. Flauzac, E. M. G. Robledo and F. Nolot . Is Mininet The Right Solution For An SDN Testbed?. 2019 IEEE Global Communications Conference (GLOBECOM). IEEE, 2019

41. Z. Fan and R. Liu . Investigation Of Machine Learning Based Network Traffic Classification. 2017 International Symposium on Wireless Communication Systems (ISWCS), 2017, pp. 1-6.

42. A. Alzubi . Learner Performance Prediction In The E-learning Platform Using The Optimized Deep Long Short-term Memory Classifier. International Journal of Wavelets, Multiresolution and Information Processing 20, 2 (2022), pp. 29. Id/No 2150051.

43. A. D. Caigny, K. Coussement and K. W. D. Bock . A New Hybrid Classification Algorithm For Customer Churn Prediction Based On Logistic Regression And Decision Trees. European Journal of Operational Research 269, 2, pp. 760--772, 2018

44. D. Arthur . A Hybrid Quantum-classical Neural Network Architecture For Binary Classification. arXiv preprint arXiv:2201.01820, 2022

45. M. Hosseinpour, S.. Ghaemi, S. Khanmohammadi and S. Daneshvar . A Hybrid High-Order

Type-2 FCM Improved Random Forest Classification Method For Breast Cancer Risk Assessment. Applied Mathematics and Computation, 2022, 424, 127038

46. D. Comaneci and C. Dobre. Securing Networks Using SDN and Machine Learning. In 2018 IEEE International Conference on Computational Science and Engineering (CSE), IEEE, 2018

47. M. H. Khairi, S. H. Ariffin, N. A. Latiff and K. M. Yusof. Generation And Collection Of Data For Normal And Conflicting Flows In Software-defined Network Flow Table. Indonesian J. Electr. Eng. Comput. Sci, 22(1), 307 (2021).

48. P. Melin, I. Miramontes and G. P. Arechiga . Nature-inspired Optimization Of Type-2 Fuzzy

Neural Hybrid Models For Classification In Medical Diagnosis. Cham: Springer, 2022

49. I. K. Nti, A. F. Adekoya and B. A. Weyori . A Comprehensive Evaluation Of Ensemble Learning For Stock-market Prediction. Journal of Big Data 7, 1 (2020), pp. 1--40

50. X. Dong, Z. Yu, W. Cao, Y. Shi and Q. Ma . A Survey On Ensemble Learning. Frontiers of

Computer Science 14, 2020, pp. 241--258

51. Z. Fang, Y. Wang, L. Peng and H. Hong. A Comparative Study Of Heterogeneous Ensemble-learning Techniques For Landslide Susceptibility Mapping. International Journal of Geographical Information Science 35, 2, 2020, pp. 321--347

52. T. Kanazawa and T. Wettig. Complete Random Matrix Classification Of SYK Models With Ð=0, 1 And 2 Supersymmetry. JHEP 09, 2017, pp. 050

53. O. Y. Bakhteev and V. V. Strijov. Comprehensive Analysis Of Gradient-based Hyperparameter Optimization Algorithms. Annals of Operations Research 289, 1, 2017, pp. 51--65

54. F. A. Anifowose, J. Labadin and A. Abdulraheem. Ensemble Machine Learning: An Untapped Modeling Paradigm For Petroleum Reservoir Characterization. Journal of Petroleum Science and Engineering 151, 2017, pp. 480--487

55. F. Anifowose, C. Ayadiuno and F. Reshedan. Feature Selection Based Hybrid Machine Learning Approach To Formation Cementation Factor Prediction. SPE Kuwait Oil and Gas Show and Conference, 2019

56. R. Garg. A Primer to Ensemble Learning--Bagging and Boosting. Analytics India Magazine, 2018

57. F. Liu, G. Kibalya, S. K. Santhosh and P. Zhang. Challenges of Traditional Networks and Development of Programmable Networks. Software-defined Internet Of Everything, Springer, 2021, 37-61

58. G. Shah, R. Valiente, N. Gupta, S. M. O.Gani, B. Toghi, Y. P. Fallah and S. D. Gupta . Real-time Hardware-in-the-loop Emulation Framework For Dsrc-based Connected Vehicle Applications. IEEE 2nd 21Connected and Automated Vehicles Symposium (CAVS), 2019, pp. 1--6.

59. D. Dholakiya, T. Kshirsagar, and A. Nayak. Survey Of Mininet Challenges, Opportunities, And Application In Software-defined Network (SDN). Information and Communication Technology for Intelligent Systems: Proceedings of ICTIS 2020, Volume 2, 2021, pp. 213--221.

60. G. D. Lena, A. Tomassilli, D. Saucez, F. Giroire, T. Turletti and C. Lac . Mininet On Steroids: Exploiting The Cloud For Mininet Performance. IEEE Xplore, 2019

61. M. Abu-Alhaija, N. M. Turab and A. Hamza. Extensive Study of Cloud Computing Technologies, Threats and Solutions Prospective. Computer Systems Science & Engineering , Vol. 41, No. 1, 2022

62. S. Din, A. Paul and A. Rehman . 5G-enabled Hierarchical Architecture For Software-defined Intelligent Transportation System. Computer Networks 150 Elsevier, 2019, pp. 81 - 89.

63. Z. O. U. Tengkuan, W. Yuying, W. U. Chengrong . Review Of Network Background-

Traffic Classification And Identification. Journal of Computer Applications , Vol. 39, No. 3 p. 802, 2019

64. B. Almadani, A. Beg and A. Mahmoud . DSF: A Distributed Sdn Control Plane Framework For The East/West Interface", IEEE Access 9, 2021, pp. 26735--26754.

65. Lee, C. Self-Detecting Traffic Interference Control For Multi-Zone Services Under 5G-Based Cellular Networks. Sensors, 2021, 21

66. K. S. Sahoo, P. Mishra, M. Tiwary, S. Ramasubbareddy, B. Balusamy and A. H. Gandomi . Improving End-users Utility In Software-defined Wide Area Network Systems. IEEE Transactions On Network And Service Management, 17(2), 696-707, 2019

67. Z. Yang, Y. Cui, B. Li, Y. Liu and Y. Xu . Software-defined Wide Area Network (SD-WAN): Architecture, Advances And Opportunities. 28th International Conference on Computer Communication and Networks (ICCCN), 2019, pp. 1 - 9.

68. X. Yuan, Z. Liu, Y. Park, H. Hu, and H. Li . Teaching SDN Security Using Hands-on Labs in CloudLab. Journal of The Colloquium for Information Systems Security Education vol. 7, no. 1, 2020, pp. 6--6.

69. K. Bao, J. D. Matyjas, F. Hu and S. Kumar . Intelligent Software-Defined Mesh Networks With LinkFailure Adaptive Traffic Balancing. IEEE Transactions on Cognitive Communications and Networking 4, 2, 2018, pp. 266--276.

70. F. Pakzad, M. Portmann, W. L. Tan and J. Indulska . Efficient Topology Discovery In Software- defined Networks. 8th International Conference On Signal Processing And Communication Systems (ICSPCS), 2014, pp. 1--8.

71. D. Hasan and M. Othman "Efficient Topology Discovery In Software-defined Networks: Revisited", Procedia computer science 116, pp. 539--547, 2017

72. S. Thakur and S. K. Jha . Cloud Computing And Its Emerging Trends On Big Data Analytics. 4th International Conference on Electronics and Sustainable Communication Systems (ICESC), 2023, pp. 1159--1164.

73. M. Kansal, P. Singh, M. K. Singh and S. Varshney . A Systematic Study of Services and Security Model in Cloud Computing: A Brief Overview. Convergence of Cloud Computing, AI, and Agricultural Science, 2023, pp. 1--16.

74. W. Hassan, T. Chou, X. Li, P. Appiah-Kubi and O. Tamer . Latest Trends, Challenges And Solutions In Security In The Era Of Cloud Computing And Software Defined Networks. Int J Inf & Commun Technol ISSN 2252, 8776, 2019, pp. 8776.

75. P. Yang, N. Xiong and J. Ren . Data security and privacy protection for cloud storage: A survey. IEEE Access 8, 2020

76. H. S. Haji, M. R S. Zeebaree, H. R. Saeed, Y. S. Ameen, M. H. Shukur, N. Omar, M. A. M.

Sadeeq, S. Z. Ageed, M. I. Ibrahim, M. H. Yasin. Comparison of software defined networking with traditional networking. Asian Journal of Research in Computer Science , Vol. 9, No. 2 p. 1-18, 2021

77. H. S. Haji, M. R S. Zeebaree, H. R. Saeed, Y. S. Ameen, M. H. Shukur, N. Omar, M. A. M. Sadeeq, S. Z. Ageed, M. I. Ibrahim, M. H. Yasin. Comparison of software defined networking with traditional networking. Asian Journal of Research in Computer Science , Vol. 9, No. 2 p. 1-18, 2021

78. S. Ahmad, A. H. Mir. Scalability, consistency, reliability and security in SDN controllers: a survey of diverse SDN controllers. Journal of Network and Systems Management. Vol. 29 Springer p. 1-59, 2021

79. K. Park, S. Sung, H. Kim, J. Jung. Technology trends and challenges in SDN and service assurance for end-to-end network slicing Computer Networks Elsevier p. 109908, 2023

Chapter Two
Literature Review

2.1 Conceptual Review

The focus of this chapter is on the review of relevant literature on the variables that are related to the research review. The following concerns have been examined in this regard: computer network and its quality of service -latency, bandwidth, throughput, and jitter as well as ensemble machine learning models. In addition, this chapter explains the various concepts, theories, and empirical reviews around the performance of stack ensemble models using datasets from the impact of

Background-Traffic and Bandwidth-Limit on Software-Defined Networking. Software-Defined Networking is programmable computer networking that alleviates the complexities of traditional computer networks due to the demand for adding more devices and responsive applications to the computer network. The stacked model performance gives the model that could be trusted to predict the factors that could impede the performance of Software-Defined Networking for advanced functionality of computer networks.

Computer networks, for example, Personal Area Network (PAN), Local Area Network (LAN), Metropolitan Area Network (MAN), and Wide Area Network (WAN) comprises a network of interconnected computers and various devices capable of seamless communication with one another, share resources, and exchange data. Computer networks vary in size and complexity, from small local networks within a home or office to large global networks like the Internet. Computer networks are a fundamental part of modern computing and play a crucial role in enabling communication and data sharing among devices and users. The programmability of computer networks like Software-Defined Networking (SDN) introduces massive advancement in a computer network that offsets the complexity of a traditional computer network. Researchers are applying machine learning to improve the responsiveness of advanced computer networks[1].

2.1.1 Computer Network Reachability and Performance

Computer network reachability and performance are critical aspects of network design and operation. They refer to the ability of devices on a network to communicate with each other and the quality of that communication. Network reachability, also known as connectivity, is the ability of devices on a network to communicate. It encompasses various aspects, including Physical Connectivity, Logical Connectivity, Firewalls and Access Control (AC), Domain Name System

(DNS), and Network Address Translation (NAT). Network performance relates to data transmission speed, reliability, and efficiency. Several factors influence network performance. These include bandwidth, which is the capacity of the network to transmit data. Higher bandwidth generally leads to faster data transfer. Latency also influences network performance. Latency is the delay that data experiences when traveling through a network. Lower latency is essential for real-time applications like video conferencing and online gaming. Packet loss influences network performance. Packet loss is rated in percentage and is describe as the situation where data packets transmitted over a network do not reach their intended destination. Reducing packet loss is crucial for reliable communication[2]. Jitter also influences network performance. Jitter is the variability in the delay of data packets. Consistent and low jitter is necessary for applications like Voice over IP (VoIP). Quality of Service (QoS) influences network performance. QoS mechanisms prioritize certain traffic types to ensure critical applications get the necessary resources. Network topology influences network performance. A network's arrangement of physical and logical components is called network topology. This arrangement can have an impact on the performance of the network. Star, mesh, and ring topologies have different characteristics. The quality and capacity of routers, switches, and other network devices affect performance. Congestion Control influences network performance. Techniques like congestion avoidance and congestion control algorithms help prevent network congestion, which can degrade performance. Ensuring network reachability and performance requires careful planning, design, ongoing monitoring, and optimizing the network for changes in demand or potential issues. Network researchers use Load Balancing (LB), Content Delivery Networks (CDNs), and efficient routing protocols to ensure network reachability and performance goals[3].

2.1.2 Latency

Latency in computer networking is the time it takes for a packet of data to travel from one device to

another device on a network. Latency is measured in milliseconds (ms). It is affected by various factors, such as the distance between the sender (source) device and receiver (destination) device, the speed of the network connection, and the amount of traffic on the network. These two end devices can be two personal computers or servers. High latency usually causes delays and slowdowns in network communication and is one of the metrics in determining the overall performance of a network[4]. Latency is a term commonly used to refer to the time it takes for a signal to travel back and forth, also known as Round-Trip Time (RTT). Latency is the time it takes for a message to reach its recipient and for the sender to receive a reply. The lower the latency, the faster speeds and quick response times[5]. High latency creates bottlenecks in communication due to long delays, which results in a worse experience for the user, as they experience poor quality of service. Organizations that rely on real-time applications, online services, or time-sensitive transactions require low lag[6]. Minimizing latency is one of the ways of improving computer networks performance. A relatively small increase in response times can ruin a user's overall experience and reduce organization yield.

2.1.3 Jitter

Jitter is the time variation between the transmission and reception of data packets over a network. Jitter occurs primarily due to network congestion, routing issues, limited bandwidth, hardware limitations, and network interference. Jitter can occur when packets take different paths or experience varying delays due to network issues. Jitter is particularly problematic for real-time applications like VoIP and video conferencing, where consistent packet delivery is crucial. Network jitter measures the irregular arrival time of data packets at their destination, also known as the jitter score, measured in milliseconds (ms). Jitter affects online activities such as real-time communication, for example, online gaming, audio and video conference calls, IP security cameras, and more[7]. The acceptable level of jitter depends on the network quality[8,9]. Jitter can occur due to

various factors, such as network congestion, varying routing paths, limited bandwidth, or even fluctuations in network traffic[10]. Some factors resulting from network jitter include Network Congestion, Routing, Network Interference, Quality of Service (QoS) mismanagement, Network Equipment Performance, Network Protocol Limitations, Packet Loss and Retransmissions, and Wireless Networks.

Network congestion is a common challenge in data communication systems, and it can lead to performance issues and service degradation. Managing and mitigating congestion involves a combination of network design, congestion control algorithms, and traffic management strategies to ensure that data can flow efficiently through the network, even during periods of high demand. Inefficient routing paths, node misconfigurations, and hardware failures can introduce delays and jitter in network communication. Addressing these issues requires good network design, monitoring, and the implementation of best practices to maintain optimal network performance. When packets traverse different paths, variations in latency and delays arise, resulting in jitter[11]. Electromagnetic interference from nearby electronic devices, crosstalk between network cables, or signal degradation due to environmental factors introduce fluctuations and inconsistencies in packet delivery times, contributing to jitter[12]. Quality of Service (QoS) mechanisms are employed to prioritize specific types of network traffic based on their importance or requirements. However, poor QoS configurations or mismanagement of network resources due to manual network configuration result in inconsistent packet prioritization[13]. The performance and optimization of network devices play a crucial role in maintaining smooth packet transmission. Faulty or poorly configured routers, switches, or network interface cards (NICs) introduce irregularities and delays in packet forwarding or processing, causing variations in packet arrival times and contributing to jitter[14]. When packets are lost during transmission due to network congestion, errors, or faulty connections, they need to be retransmitted. This retransmission introduces delays and variations in packet arrival times, contributing to jitter. High packet loss rates or frequent retransmissions

exacerbate the effects of jitter on network performance[15]. In wireless networks, factors such as signal interference, signal strength fluctuations, and contention for the shared medium introduce jitter[16]. Interference from other devices, obstacles obstructing the signal path, or fluctuations in signal quality due to environmental conditions can cause variations in packet delivery times in wireless transmissions. By identifying and addressing these specific causes of jitter, network administrators can implement appropriate programmable solutions and optimizations to minimize impact of jitter on network and ensure more reliable and consistent network performance.

2.1.4 Throughput

Throughput is a crucial performance metric as it directly impacts the efficiency and effectiveness of various systems and processes. Organizations often strive to maximize throughput for higher productivity, faster data transfer, and better overall performance. Throughput is a critical performance metric in various fields, including Networking, Manufacturing and Production, Computer Systems, Data Processing, Transportation, and Communication Systems.

Throughput in computer networking refers to the amount of data transmitted from one location to another over a network in a given time. It is typically measured in bits per second (bps), kilobits per second (Kbps), megabits per second (Mbps), or gigabits per second (Gbps), depending on the speed of the network. Higher network throughput indicates faster data transfer speeds. Several factors can significantly impact network throughput, including bandwidth, latency, congestion, packet loss, protocol overhead, network equipment, topology, and QoS configuration. Bandwidth is the network's capacity, representing how much data can be transmitted in a specific period. A network with higher bandwidth can generally transmit more data and, therefore, has higher throughput[17]. Latency is the time it takes for data to travel from the source to the destination. High latency reduce throughput because it introduces delays in data transmission[18]. When many devices share the same

network, it can become congested, leading to a reduction in throughput as data packets compete for limited resources[19]. When data packets are lost during transmission due to network errors or congestion, it can affect the overall throughput because the lost packets need to be re-transmitted[20]. Networking protocols add overhead to the data being transmitted. This overhead reduces the effective throughput because it consumes part of the available bandwidth. The quality and capacity of networking equipment, such as routers, switches, and cables, can also impact throughput. Outdated or low-quality equipment limit the network's ability to transmit data efficiently. The network's physical and logical design influence throughput. Quality of Service (QoS) mechanisms have been used by computer network researchers to prioritize certain types of traffic over others, ensuring that critical data gets better throughput than less essential data[21]. To measure and optimize throughput in a network, network researchers often use various tools and techniques. Throughput testing tools, such as iPerf, are used to assess the data transfer rates on a network.

2.1.5 Bandwidth

Bandwidth is the capacity of a network or communication channel to transfer data, and it is a critical concept in computer networking. However, the term "bandwidth" is used in various contexts. Generally, bandwidth is used metaphorically to describe an individual or organization's capacity or ability to handle tasks or information. For example, an individual or organization's capacity would have limited "bandwidth" to take on additional work, that is, little capacity or time to carry out more tasks. In Computer Networking, bandwidth typically refers to the data transfer rate or capacity of a network or internet connection. It is usually measured in bits per second (bps), kilobits per second (Kbps), megabits per second (Mbps), or gigabits per second (Gbps). Higher bandwidth means a network can transmit more data in a given time, resulting in faster data transfer speeds. In Signal Processing, bandwidth refers to frequencies within a signal or a transmission channel[22]. For instance, in audio, it describe the range of frequencies that a speaker or audio

machine can reproduce accurately. In Electronics and Electrical Engineering, bandwidth refers to the range of frequencies that a component or system can handle[23]. For example, the bandwidth of an amplifier indicates the range of frequencies it can amplify without significant distortion. In Finance and Economics, bandwidth describes the range of values or prices within which a financial instrument or asset fluctuates[24]. It also refers to the width of a price range in technical analysis. Bandwidth in Computer Networking is the maximum data transfer rate of a network or communication channel. Some areas relating to bandwidth in Computer Networking include Data Transfer Rate, Upstream and Downstream, Latency vs. Bandwidth, Measuring Units, Shared Resources, Factors Affecting Bandwidth, Symmetric vs. Asymmetric, Scalability and Quality of Service (QoS).

In Data Transfer Rate, bandwidth represents how much data can be transmitted over a network or communication channel in a given time[25]. For example, a network with a bandwidth of 100 Mbps can transmit 100 megabits of data per second. In Upstream and Downstream, upstream bandwidth refers to the rate at which data can be sent from a user or device to the network, while downstream bandwidth is the rate at which information can be received from the network. Latency is the delay in data transmission, while bandwidth is the capacity for data transfer. A network can have high bandwidth but still experience high latency due to other factors like physical distance. Bandwidth is commonly quantified as the amount of data that can be transferred in a second, which is measured in bits per second (bps) and is often expressed in larger units such as Kbps, Mbps, Gbps, or even Tbps for high-speed connections. In shared resources, in many cases, bandwidth is a resource that is shared. In a network with multiple users, the available bandwidth is divided among the users. If one user consumes a significant portion of the available bandwidth, it can affect the performance of other users on the same network. Several factors affect the adequate bandwidth in a network, including network congestion, interference, the quality of network equipment, and the distance between devices. The actual data transfer rate might be lower than the theoretical maximum due to

these factors. For the Symmetric vs. Asymmetric, some networks offer symmetric bandwidth, where the upload and download speeds are the same for example in fiber-optic connections has capability to offer symmetrical bandwidth, that is the upload speeds are exactly the same as the download speeds[26,27]. For network scalability, network administrators often need to plan for scalability, ensuring that the available bandwidth can handle increasing data demands as more users and devices are added to the network. As for Quality of Service (QoS), in some cases, it's important to prioritize certain types of data or applications over others.

2.1.6 Background-Traffic

In computer networks, Background-Traffic refers to the non-essential or non-legitimate data that flows over the network alongside the primary, prominent, or legitimate data traffic[28]. This Background-Traffic includes various types of communication that are not time-sensitive or mission-critical[29]. Common examples of Background-Traffic in a network include Updates and Maintenance, Backup and Replication, Logging and Monitoring, Network Management, Email and Non-Real-Time Communication, File Transfers, Content Delivery, and Streaming. Software updates, patches, and system maintenance tasks are often transmitted as Background-Traffic. These operations consume network bandwidth, but they are typically scheduled during off-peak hours to minimize disruption to regular network activities. Data backup and replication processes generate significant Background-Traffic[30]. This traffic is usually scheduled to occur when network utilization is low to prevent interference with regular operations. Network devices, servers, and applications often generate log files and monitor data sent across the network. While this information is essential for troubleshooting and security, it's considered Background-Traffic because it's not part of the primary data transfer. Network management traffic, including Simple Network Management Protocol (SNMP) messages and other control traffic, is vital for monitoring and controlling network devices. It's typically low in volume compared to data traffic. Emails and

non-real-time messaging services are also considered Background-Traffic. Unlike video conferencing or real-time voice calls, email and chat messages are less time-sensitive. Web browsing generates Background-Traffic when users access websites. While this traffic is user-initiated, it is often less time-sensitive than other applications, so it's categorized as Background-Traffic in the context of network prioritization. Large file transfers or downloads that are not time-critical are often considered Background-Traffic. These transfers affect network performance, especially on shared networks, but they are usually scheduled during off-peak hours when possible. Streaming media, such as video or audio, can be a significant source of Background-Traffic. While real-time streaming is more critical, Background-Traffic includes preloading content or buffering to ensure smooth playback. Managing Background-Traffic is essential for maintaining network performance and quality of service for critical applications. Network administrators and researchers utilize quality of Service (QoS) mechanisms to prioritize and allocate bandwidth to essential applications. To guarantee that basic operations are not interrupted, it is necessary to take measures -Background-Traffic. Achieving a balanced and efficient network environment involves setting policies, shaping traffic, and classification.

2.1.7 Bandwidth-Limit

It's essential to understand the Bandwidth-Limits of a network to ensure that online activities run smoothly. Bandwidth is a term commonly used in the context of computer networks and the Internet. The term bandwidth refers to the amount at which data can be transmitted over a network or internet connection. Bandwidth is measured in bits per second (bps) and describes how much data can be transmitted over the network in a given time. Upload bandwidth is the amount of data sent from one end device to another in the network or the internet. For example, pushing a file to a server or sending an email uses upload bandwidth. Download bandwidth is the data an end device can receive from another end device in the network or the internet. When a network user streams a

video, browse a website, or retrieve a file from an end device, the network user uses download bandwidth. Bandwidth is a finite resource, and it can be limited intentionally or unintentionally for various reasons like network configuration, network congestion, data plans, technology limitations, hardware limitations, and geographic factors. Network administrators and researchers intentionally limit bandwidth for various purposes, such as ensuring fair usage among multiple users or managing network congestion. This issue can be resolved by implementing Quality of Service (QoS) settings or traffic shaping. In networks that are crowded or oversubscribed, Bandwidth-Limitations arise due to the high number of users attempting to access the network simultaneously. Mobile and home internet plans often have specific Bandwidth-Limits. Available bandwidth is also constrained by the technology used in the network. For example, older DSL connections typically have lower bandwidth than fiber-optic connections. The capabilities of networking equipment, such as routers or modems, also influence the adequate bandwidth available to devices in the network[31]. The distance between the end device and the network server affect the available bandwidth, especially in the case of satellite internet or long-distance data transmission[32].

2.1.8 Iperf

Iperf is a widely used open-source tool for measuring the maximum Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) bandwidth performance of a network. It's a command-line utility used to test the performance of computer networks by generating and measuring network traffic. Iperf is commonly used for network testing, troubleshooting, and planning. Network researchers use Iperf to measure the bandwidth between two networked devices[33]. Iperf is particularly useful in scenarios like setting up a new network, troubleshooting network performance issues, or verifying the performance of a network link. Iperf has been used to troubleshoot networks to identify network issues by providing detailed information about network performance, including

metrics such as throughput, jitter, and packet loss[34]. Iperf has been used to test if a network is providing the required service quality, particularly in environments where QoS is necessary, such as VoIP or video conferencing. In network capacity planning, Iperf has been used to determine if a network link has sufficient capacity to support the intended applications and services[35]. Iperf operates in a client-server model. One machine runs in server mode, and the other in client mode.

Server Side: On one of the machines, Iperf was run in server mode and listens for incoming connections:

shell prompt$ iperf -s

Client Side: On another machine, Iperf was run in client mode with specify server's IP address:

shell prompt$ iperf -c server_ip

Iperf generates network traffic between the client and server and reports various performance metrics, including bandwidth, jitter, and packet loss[36].

2.1.9 Wireshark

Wireshark is an invaluable tool for monitoring network performance, diagnosing network problems, and investigating security incidents[37,38] Wireshark is a popular and widely used open-source network protocol analyzer. Wireshark is available for multiple platforms, including Windows, MacOS, and Linux. It's a tool that allows one to capture and inspect the data traveling back and forth on a computer network[39]. Wireshark is highly regarded for its versatility and is typically used by network administrators, computer network researchers, and developers to troubleshoot network issues and analyze network performance. Some functions of Wireshark include packet capture, packet analysis, real-time analysis, protocol support, powerful filtering, exporting data, colorization,

graphing and customization. Wireshark has been used to capture and display data packets (data units) of various network protocols as they traverse a network interface[40]. Wireshark is a network protocol analyzer that monitors wired and wireless networks. It is a powerful tool for analyzing network traffic, capturing packets, and troubleshooting network issues. Wireshark provides a detailed and hierarchical view of captured packets, allowing researchers to inspect the contents of individual packets and understand the structure of network communication[41]. Wireshark has been used to capture and display packets in real-time, which helps monitor live network traffic as well as many network protocols, including Ethernet, IP, TCP, UDP, HTTP, DNS, SSL/TLS[42]. Wireshark has been used to analyze Voice over IP (VoIP) traffic, making it useful for troubleshooting and quality assessment of VoIP calls[43]. Captured data are saved using various formats, such as pcap, which can be opened in other network analysis tools, or as CSV for spreadsheet applications. Wireshark uses colors to highlight different types of packets, making it easier to identify issues. Wireshark also generates various graphs to visualize network statistics and allows researchers to create and apply custom display filters and protocol dissectors[44].

2.1.10 Traditional Computer Network (TCN)

Traditional Computer Network (TCN) is a common computer network. It is a set of interconnected computers and devices communicating and sharing resources. These networks vary in size and complexity, but there are several common types and components around. Standard terms used in computer networking include the following;

Local Area Network (LAN) is a network that typically covers a small geographical area, like a home, office, or campus. It's used for connecting devices within a limited area. Computers and devices in a LAN share resources like printers and files.

Wide Area Network (WAN) covers a larger geographical area, often connecting LANs in different

locations. The Internet itself is the most prominent example of a WAN.

Wireless LAN, also known as Wi-Fi or WLAN, is a type of LAN that uses wireless connections, typically over radio waves, to connect devices. Wi-Fi is common in homes and businesses. Ethernet Networks use ethernet, which is a standard technology for wired LANs. It uses physical cables to connect devices in a network, and it can offer high-speed, reliable connections.

The **Internet** is the global interconnection of networks that connects billions of devices worldwide. It uses various technologies, including fiber optics, satellite, and undersea cables. Routers are devices that connect different networks. In a home network, it connects the local network to the Internet. In more extensive networks, it directs data between LANs and the Internet.

Switches are used within LANs to direct data between devices on the same network. They operate at the data link layer and are more efficient than hubs.

Firewalls are a crucial security measure to safeguard networks and systems against unauthorized access and cyber threats by controlling the incoming and outgoing network traffic. They can be implemented as hardware devices or software applications.

A **MODEM** is a Modulator-Demodulator that converts digital data from a computer into analog signals for transmission over analog networks (like telephone lines) or vice versa. In many cases, modems are integrated into routers.

Network Protocols are the rules and conventions that govern how data is transmitted and received on a network. Common examples include TCP/IP (used on the Internet), HTTP (for web communication), and SMTP (for email).

An **IP address** is the logical address assigned to a computer network node, and it is used to identify network hosts. IPv4 and IPv6 are typical IP addressing schemes. Domain Name System (DNS) translates human-friendly domain names (like www.example.com) into IP addresses.

LAN cables (e.g., Ethernet cables) connect devices in a wired network. They are made of copper or fiber optics.

A Wireless **Access Point** (AP) is an access point that provides a wireless connection for devices to connect to the LAN or the Internet.

In networked systems, a "**client**" is typically a device or application that requests services or resources, and a "**server**" provides those services or resources.

An **intranet** is a private network that is confined to an organization and is used for internal communication, often built using ethernet technologies.

An **extranet** is similar to an intranet but allows for limited access by external parties, such as business partners or customers.

Virtual Private Network (VPN) is a technology that provides secure and encrypted communication over public networks such as the Internet. It is often used for remote access to a private network.

2.1.11 Software-Defined Networking

Software-Defined Networking (SDN) is an innovation in network management and configuration that separates the control plane from the data plane of network devices, making networks more flexible, scalable, and programmable. SDN is designed to address the limitations and inflexibility of traditional network architectures. SDN can make network management more agile, cost-effective, and adaptable to the dynamic needs of modern applications and services. SDN has numerous applications, from data center network management to Wide Area Network (WAN) optimization and even in emerging technologies like 5G networks. Components and concepts associated with

SDN include control plane and data plane, SDN controller, southbound APIs, northbound APIs, flow tables, programmability, virtualization, traffic engineering, and optimization.

In traditional network devices (such as routers and switches), the control plane, responsible for making decisions about forwarding traffic, is tightly integrated with the data plane, which is responsible for moving data packets. SDN decouples these two planes[45]. The brain of an SDN network is the SDN controller. SDN controller is a centralized software entity that acts as a single control point for the entire network. Network administrators communicate with the SDN controller through an API to configure the network, define traffic policies, and decide how data should be forwarded. Southbound APIs are the interfaces between the SDN controller and the network devices in the data plane. They allow the controller to communicate with network devices to configure them and provide instructions on how to forward traffic. Common southbound APIs include OpenFlow. Northbound APIs allow applications and services to communicate with the SDN controller. Network services, applications, and management tools can request and receive information from the controller through northbound APIs. In the data plane, network devices (e.g., switches) have flow tables that dictate how incoming packets are processed. These flow tables are typically programmed by the SDN controller based on network policies. SDN networks are highly programmable, allowing administrators to create and modify network configurations dynamically. This flexibility is crucial for adapting to changing traffic patterns, security requirements, and other network demands. SDNs are combined with network virtualization technologies like network overlays (e.g., VXLAN, NVGRE) to create logical network segments that operate independently of the underlying physical network, providing flexibility and isolation. SDN enables more efficient use of network resources by allowing dynamic traffic engineering. The network can automatically route traffic based on real-time conditions, optimizing for latency, bandwidth, and cost. With the centralization of network control, network administrators have the ability to manage the entire network using a single interface., which simplifies network administration and reduces the risk of configuration errors[46].

Technologies, such as OpenFlow, are typically based on open standards, making it easier for different vendors' hardware and software to inter-operate.

2.1.12 Combined Machine Learning Models

Ensemble learning refers to the process of combining multiple machine learning models to create a more robust predictive model. This technique combines various models to develop a more comprehensive and accurate model, which is known as ensemble modeling[47]. By doing so, ensemble modeling creates a more precise predictive model compared to any individual model. This technique is widely used in machine-learning competitions and real-world applications because it improve the performance of machine-learning algorithms[48]. Several methods for combining machine learning models are highlighted in the following section.

2.1.12.1 Ensemble Methods

Bagging (Bootstrap Aggregating) is one of the methods used in ensemble learning. This method concerns training multiple instances of the same model on different subsets of the training data and then averaging or taking a majority vote to make predictions[49]. Random Forest (RF) is a famous example of a bagging algorithm. Boosting is also one of the methods used in ensemble learning. Boosting focuses on combining weak models to create a robust model. Algorithms like AdaBoost, Gradient Boosting, and XGBoost iteratively train models, giving more weight to misclassified samples in each iteration[50,51]. Stacking is one of the methods used in ensemble learning. Stacking combines the predictions of multiple base models by training a meta-model that takes the outputs of the base models as inputs[52]. The concept can be visualized as a two-tiered group (Figure 2.1).

K-Nearest Neighbors (KNN) is a simple, instance-based learning algorithm used for classification and regression tasks. The basic idea behind KNN is to classify a data point based on how its

neighbors are classified. For classification, it calculates the distances between the test data and all the training data points, and assigns the class that is most common among the k-nearest neighbors (where k is a user-defined constant). For regression, it predicts the average value of the nearest neighbors. KNN is easy to implement and understand, but it can be computationally intensive as it requires calculating the distance of each data point to all other points in the dataset, especially for large datasets. It is also sensitive to the choice of k and the scaling of the data.

Support Vector Machine with Radial Basis Function Kernel (SVM_RBF). Support Vector Machine (SVM) with Radial Basis Function (RBF) Kernel is a powerful classification algorithm that constructs a hyperplane or set of hyperplanes in a high-dimensional space to separate different classes. The RBF kernel is a popular choice for SVM as it can handle non-linear relationships by transforming the input space into a higher-dimensional space where a linear separator can be used. SVMs are effective in high-dimensional spaces and are robust against overfitting, especially in cases where the number of dimensions exceeds the number of samples. However, they can be memory-intensive and require careful selection of parameters such as the penalty parameter C and the kernel coefficient gamma.

Decision Tree (DT) is a non-parametric supervised learning method used for classification and regression tasks. It splits the data into subsets based on the value of input features, creating a tree structure where each node represents a feature, each branch represents a decision rule, and each leaf node represents an outcome. Decision Trees are easy to visualize and interpret, making them popular for understanding and interpreting model predictions. They can handle both numerical and categorical data and require little data pre-processing. However, they are prone to overfitting, especially when the tree is deep, and are sensitive to noisy data, which can result in complex trees that do not generalize well.

Random Forest (RF) is an ensemble learning method that combines multiple decision trees to create a more robust and accurate model. It constructs a multitude of decision trees during training

and outputs the mode of the classes for classification or the mean prediction for regression of the individual trees. The idea is to reduce the overfitting typically seen in individual decision trees by averaging their results. Random Forests handle large datasets well and are robust to noise, making them suitable for a wide range of tasks. However, they are less interpretable than single decision trees and can be computationally intensive, especially with a large number of trees and high-dimensional data.

Multi-Layer Perceptron (MLP) is a class of feedforward artificial neural network that consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Each node (except for the input nodes) uses a non-linear activation function, and MLP uses backpropagation for training the network. MLPs are capable of modeling complex, non-linear relationships and can learn representations from data, making them highly flexible. However, they require a large amount of data and computational power to train effectively. Additionally, MLPs are sensitive to the choice of hyperparameters and the initialization of weights, and they can overfit if not properly regularize.

Figure 2.1: Stacking model
Source[53]

2.1.12.2 Voting

In ensemble learning methods such as Random Forest, the approach involves training multiple models independently and then combining their predictions to enhance overall performance and accuracy. Specifically, Random Forest constructs numerous decision trees during training and aggregates their predictions through a voting mechanism. Each decision tree makes an individual prediction, and the final outcome is determined by the majority vote among all the trees. This voting process ensures that the collective decision is more robust and less prone to errors than any single model's prediction. Moreover, ensemble learning can also extend beyond a single type of model by incorporating different algorithms like regression models, Support Vector Machines (SVMs), and decision trees. In this case, the predictions from these diverse models are combined, and the most frequent prediction among them is chosen as the final decision. This strategy leverages the strengths of various models, reducing the likelihood of overfitting and increasing the predictive power and generalization capability of the ensemble[54].

2.1.12.3 Averaging and Weighted Averaging

Models are combined by averaging their predictions, often used with regression models[55]. Averaging is a technique used in machine learning and statistical modeling where multiple model predictions are combined to produce a final prediction. This approach is particularly common in regression tasks. By averaging the predictions of several models, the overall prediction is often more robust and less sensitive to the idiosyncrasies of individual models. The underlying principle is that while individual models might have specific biases or errors, these can be mitigated by combining multiple models, as the errors of different models may cancel each other out[56]. This method can significantly improve predictive performance and reliability, making it a valuable tool in economic forecasting and other fields that require precise prediction.

Weighted averaging is a refinement of the simple averaging technique. Instead of treating all models equally, weighted averaging assigns different weights to the predictions of different models. This allows data analysts to prioritize certain models that they believe to be more accurate or reliable based on prior performance or other criteria. By adjusting the weights, the final prediction can be tailored to reflect the relative importance of each model's output. This method provides greater flexibility and can lead to better performance than simple averaging, particularly when some models are known to perform better under specific conditions. This approach enhance the overall predictive power by leveraging the strengths of the most reliable models while still incorporating the diverse perspectives of multiple models[57].

2.1.12.4 Model Stacking

Model stacking is an advanced ensemble technique in machine learning that involves training multiple diverse models and using their predictions as input features for a meta-model. This method

leverages the strengths of different base models, aiming to improve overall predictive performance by combining their outputs optimally.

In the stacking process, the initial step involves training several base models, each possibly using different algorithms or training data subsets. These base models could include a variety of machine learning techniques such as decision trees, support vector machines, neural networks, or any other algorithm. The key idea is that each model captures different aspects of the data, and their combined predictions can provide a more robust and accurate output.

Once the base models are trained, their predictions are then used as new features to train a meta-model. This meta-model, also known as a level-1 model, is designed to learn the best way to combine the base models' predictions. The meta-model takes these predictions and learns from them to produce the final prediction. The effectiveness of stacking lies in the meta-model's ability to mitigate the weaknesses of individual base models by learning the optimal combination strategy. For instance, a stacking ensemble model was employed to forecast photovoltaic power generation, demonstrating how stacking can be used to improve the accuracy of predictions by combining the outputs of various base models[58]. Similarly, boosted C5.0 decision tree algorithm with a penalty factor has been used for breast cancer diagnosis, illustrating the practical applications of stacking in different domains[59]. Exploring the ensemble learning method using stacking, comparing the performance of different base learners and highlighting how stacking can lead to better predictive performance than individual models suggest that the meta-model's ability to integrate diverse predictions plays a crucial role in the success of stacking[60]. Their findings suggest that the meta-model's ability to integrate diverse predictions plays a crucial role in the success of stacking.

2.1.12.5 Bayesian Model Averaging

Bayesian Model Averaging (BMA) is a sophisticated technique that merges predictions from

multiple models by leveraging a probabilistic framework. Unlike traditional methods that select a single best model, BMA acknowledges the inherent uncertainties and variabilities across different models. It operates on the principle that all models considered have some degree of validity and thus should contribute to the final prediction in proportion to their credibility. This approach allows BMA to produce more robust and reliable predictions by integrating the strengths and insights from various models, rather than relying on one potentially flawed or limited model.

BMA calculates a weighted average of predictions from the models in question[61]. These weights are determined based on the uncertainties associated with each model, typically derived from their posterior probabilities. This probabilistic weighting ensures that models which have greater evidence supporting their accuracy and reliability are given more influence in the final averaged prediction. Consequently, BMA mitigates the risk of overconfidence in any single model and reduces the impact of model-specific errors, leading to more nuanced and dependable predictions.

The core advantage of BMA lies in its ability to formally incorporate model uncertainty into the predictive process. By considering the range of possible models and their associated uncertainties, BMA provides a comprehensive view of the prediction landscape. This is particularly valuable in complex fields such as hydrology, where the interactions between various environmental factors can be intricate and multifaceted. Through BMA, hydrologists and other practitioners can achieve more accurate and credible predictions, enhancing their decision-making processes and ultimately leading to better-informed management and policy decisions.

2.1.12.6 Hybrid Models

Hybrid models in machine learning integrate multiple model types to leverage their individual strengths and mitigate their weaknesses[62]. For instance, a hybrid model might combine a Deep Neural Network (DNN) with a decision tree-based model to achieve better performance. Deep

neural networks excel at capturing complex, non-linear relationships in data due to their deep layers and high capacity for learning from large datasets. However, they can be computationally expensive and prone to overfitting if not properly regularized. On the other hand, decision tree-based models, like XGBoost and Random Forests (RF), are generally faster to train and interpret, and they often perform well on structured data[63]. By combining these two types of models, researchers can create hybrid models that are both powerful and efficient.

In the context of water quality prediction, two specific hybrid models: CEEMDAN-XGBoost and CEEMDAN-RF has been used to enhance prediction accuracy[64]. The Complete Ensemble Empirical Mode Decomposition with Adaptive Noise (CEEMDAN) method is used to decompose the time series data into intrinsic mode functions (IMFs), which helps in capturing the essential patterns in the data more effectively. When these decomposed components are fed into XGBoost and RF models, the prediction performance improves significantly compared to using XGBoost and RF alone. This approach illustrates how hybrid models can enhance predictive accuracy by combining the strengths of different machine learning techniques[65]. Hybrid models has been explored in time-sensitive networking-driven deterministic low-latency communication for real-time telemedicine and e-health services[66]. The integration of different machine learning models in this domain ensures that the communication systems can meet the stringent latency requirements necessary for real-time medical applications. This blend of models helps in managing the trade-off between speed and accuracy, which is crucial in time-sensitive applications.

Similarly, hybrid maximum likelihood inference has been explored for stochastic block models[67]. This approach combines different statistical and machine learning methods to enhance the inference process in complex network data. By utilizing a hybrid framework, the researchers were able to achieve more robust and accurate inferences, demonstrating the versatility and effectiveness of hybrid models across various applications.

The use of hybrid models represents a significant advancement in machine learning. By combining the unique advantages of different models, researchers can achieve better performance. The flexibility and improved accuracy offered by hybrid models make them a valuable tool in the ever-evolving landscape of machine learning and data analysis.

2.1.12.7 Cascade Models

In cascade models, a sequence of multiple models is trained and applied in a specific order to make predictions. Each model in the cascade serves a particular purpose and builds upon the output of the previous models. For instance, the first model in the cascade might be designed to identify and eliminate obvious negative cases, significantly reducing the number of instances that need to be processed by subsequent models. This initial model acts as a filter, ensuring that only more complex or ambiguous cases proceed to the next stage. The subsequent models then focus on classifying the remaining cases with higher accuracy and specificity. By doing so, the computational load is distributed more efficiently, and the overall performance of the system can be improved. This approach is particularly beneficial in scenarios where the dataset is imbalanced, meaning that one class significantly outnumbers the other(s). In such cases, the initial model can help reduce the number of majority class instances that need to be handled by the later, more sophisticated models, allowing these models to focus on the minority class instances that are often harder to classify correctly, for example in a study where iQSP was developed for predicting and analyzing quorum sensing peptides[68]. In their approach, a sequence-based method using informative physicochemical properties was employed to create a predictive model. By leveraging cascade models, they could effectively manage and analyze imbalanced datasets, ensuring that the predictive performance remained robust even when faced with a disproportionate number of negative cases compared to positive ones. This technique not only enhances the efficiency of the predictive model but also improves its accuracy and reliability in handling real-world data scenarios.

2.1.12.8 Sequential Models

Combining machine learning models that operate sequentially can be beneficial in certain cases, for instance, using a time series forecasting model to predict future values and then a classification model to make decisions based on those forecasts[69].

The selection of the appropriate ML method for a specific problem depends on the data's characteristics and the available computational resources[70]. Although combining models can improve performance, it also increases complexity, and there is a risk of overfitting the ensemble[71]. Therefore, proper validation and tuning are crucial when combining machine learning models to ensure that the best results are achieved.

2.2 Methodological Review

The methodological review of the assessment of integrated machine learning models focuses on understanding how these models are applied to evaluate the influence of background traffic and bandwidth limitations on the performance of Software-Defined Networking (SDN). This involves a comprehensive examination of various machine learning techniques and integration of machine learning models. The review covers data collection methods, where real-time traffic data and network performance metrics are gathered, and pre-processing steps like normalization and feature selection. It also delves into the design of experiments to simulate different network conditions, including varying levels of background traffic and bandwidth constraints. The review evaluates the model training and validation processes, using techniques like cross-validation and performance metrics such as accuracy, latency, throughput, and jitter. Finally, it discusses the interpretation of results, comparing the effectiveness of different machine learning models in predicting and optimizing SDN performance under varying traffic and bandwidth scenarios, highlighting the

strengths and limitations of each approach.

2.2.1 Traditional Computer Network and Software-Defined Networking Architectural Frameworks

Computer Network and Software-Defined Networking (SDN) are crucial technologies in the field of networking[72]. They are guided by architectural frameworks that provide a structured approach to designing and managing networks. Outlined below are the various architectural frameworks of computer networks.

1. Open Systems Interconnection Model (OSI Model): The OSI model is a conceptual framework that standardizes computer communication system into seven distinct layers. These layers, from the lowest to the highest, are Physical, Data Link, Network, Transport, Session, Presentation, and Application[73]. The OSI model provides an apparent reference for understanding how different networking protocols and technologies interact[74].

2. TCP/IP Model: The TCP/IP model is another reference framework that's often used in the context of the Internet and modern networking[75]. It's a four-layer model and includes the Network Interface (equivalent to the OSI Data Link and Physical layers), Internet (equivalent to the OSI Network layer), Transport (equivalent to OSI Transport layer), and Application (equivalent to OSI Session, Presentation, and Application layers). This model is more closely aligned with the structure of the Internet.

3. SDN Architectural Framework: Software-Defined Networking (SDN) is a modern approach to network management that separates the control plane from the data plane[76,77]. The SDN architectural framework typically consists of three key components

a. Application Layer: This is where network applications and services reside. These applications interact with the SDN controller to define network behavior and policies.

b. Control Layer: The control plane is responsible for network management, which includes decision-making, traffic engineering, and configuring network devices. The SDN controller plays a central role in this layer.

c. Infrastructure Layer: This includes network devices (switches, routers, etc.) that forward data packets. In an SDN architecture, these devices are typically simple data plane devices that follow instructions provided by the control plane.

4. SDN Controllers: In SDN, the SDN controller is a critical component. It's responsible for translating high-level network policies set by the applications into low-level instructions that configure the network devices in the infrastructure layer[78,79].

5. Network Functions Virtualization (NFV): This framework focuses on virtualizing network functions traditionally performed by dedicated hardware appliances. NFV aims to replace or supplement these physical devices with virtualized counterparts running on standard server hardware[80].

6. Cloud Networking Architectural Framework: As cloud computing has become increasingly prevalent, networking has also evolved to cater to cloud environments. This framework includes concepts like virtual networks, load balancers, and security groups designed to meet the unique needs of cloud-based applications.

7. Data Center Network Architectural Framework: This framework deals with the design and management of data center networks. It includes technologies like Ethernet fabrics, virtual LANs (VLANs), and network virtualization[81].

8. Internet of Things (IoT) Networking Architectural Framework: IoT introduces unique challenges due to the massive number of devices and their diverse communication requirements. IoT frameworks consider how to connect, manage, and secure these devices efficiently. These architectural frameworks provide a structured way to think about and design computer networks

and SDN systems[82]. They help network engineers and architects make informed decisions about technology selection, configuration, and deployment. Software-Defined Networking (SDN) is an innovative approach to network management that allows for the centralized control of network infrastructure through software. SDN separates the control plane from the data plane, enabling more agile and programmable networks[83].

2.2.2 Components of SDN

The essential methods and components of SDN include

1. Controller: The central component of SDN is the SDN controller, and it responsible for deciding how data traffic should be forwarded throughout the network. The controller communicates with network devices and applications to enforce network policies[84].

2. Southbound APIs: These interfaces allow the SDN controller to communicate with network devices, such as switches and routers. Examples of southbound APIs include OpenFlow, NETCONF, and RESTful APIs.

3. Northbound APIs: These interfaces enable communication between the SDN controller and higher-level applications or network services. Northbound APIs allow applications to request specific network services or policies from the controller[85]. These APIs are essential for integrating SDN into existing network management systems.

4. Data Plane: The data plane, or forwarding plane, is accountable for forwarding data packets through the network devices. In SDN, the control plane (controller) makes decisions about how traffic should be delivered, and these decisions are then implemented in the data plane.

5. Switches and Routers: In SDN, network devices like switches and routers are considered "dumb" because they rely on instructions from the SDN controller to determine how to forward traffic[86].

These devices use southbound APIs to communicate with the controller and receive forwarding instructions.

6. Flow Table: Network devices typically contain flow tables, which are used to store forwarding rules. The flow table is programmed by the SDN controller and specifies how incoming packets should be forwarded based on criteria like source and destination addresses[87].

7. Network Applications: SDN allows for the development of network applications that can leverage the capabilities of the SDN controller to provide various network services. Examples of such applications include load balancers, firewalls, and traffic optimization tools.

The general process of SDN operation involves Network Discovery, Network Management, Dynamic Traffic Routing and. Integration with Network Applications[88].

1. Network Discovery: The SDN controller discovers the network topology, including the connected devices and their capabilities.

2. Network Management: The administrator defines network policies and configurations through the SDN controller, which then enforces these policies on the network devices.

3. Dynamic Traffic Routing: The SDN controller dynamically adapts to changing network conditions, optimizing traffic routing and ensuring network resources are used efficiently.

4. Integration with Network Applications: SDN allows for the development and integration of network applications that can provide additional services or automation.

Overall, SDN simplifies network management, increases network agility, and provides greater flexibility in responding to changing traffic patterns and requirements. It’s particularly valuable in data centers, cloud computing, and large-scale network environments.

2.2.3 Machine Learning Models

Machine learning models are computational algorithms or systems designed to learn patterns, make predictions, and improve their performance over time based on data. These models are a fundamental part of machine learning and are used in various applications like speech recognition, image processing, recommendation systems, and autonomous vehicles. The methods used to create and train machine learning models vary depending on the specific problem and the type of model being used[89].

In machine learning, data collection is crucial, and the first step is to gather relevant data. High-quality, clean, and representative data is essential for training a machine learning model[90]. The data usually includes the input features (attributes) and the target variable (the variable that needs to be predicted or classified). Data often needs cleaning, transformation, and normalization to make it suitable for training, which may involve handling missing values, scaling features, and encoding categorical variables. Feature selection and engineering are done in building machine learning models. Feature selection involves choosing the most relevant features contributing to the model's performance. Feature engineering, on the other hand, consists of creating new features from the existing dataset to improve the model's ability to learn[91]. Splitting data is carried out in building machine learning models.

Data is typically divided into two subsets: a training set for model training, a validation set for hyperparameter tuning and model selection, and a test set to evaluate the model's performance[92,93]. The choice of the machine learning model depends on the type of problem. Typical machine learning models include linear regression, support vector machines, decision trees, neural networks, and more. The model selection depends on factors like the problem's nature and complexity. In training the model, the model learns from the training data to optimize its parameters or coefficients to minimize the difference between its predictions and the actual target values. The specific training algorithm used depends on the chosen model. Machine learning models usually have

hyperparameters that are not learned from the data but must be set before training.

Grid or randomized search is used to find the best combination of hyperparameters that results in the best model performance[94]. The model's performance is evaluated on the validation set, which helps in assessing how well the model is doing and making necessary adjustments. The final model is evaluated on an independent test dataset to assess its generalization performance. This test is an important step to guarantee that the model does not overfit the training data[95,96]. If the model performs well, it is deployed to predict new, unseen data. Deployment can be a web application, API, or integration into a more extensive system. Machine learning models are continuously monitored for performance drift and updated as needed to adapt to changes in the data distribution or other factors[97]. Understanding the model's predictions and making them interpretable is crucial for many applications. Various techniques, such as feature importance analysis and model-specific interpretation methods, can be applied for this purpose.

2.2.3.1 Ensemble Machine Learning

Ensemble learning is a machine learning technique combining multiple models to improve prediction accuracy and reduce the risk of overfitting. It's based on the idea that combining various models can often lead to better results than individual models. Ensemble methods are commonly utilized in diverse machine learning applications[98]. Ensemble methods are categorized into several different approaches:

1. Bagging is also known as Bootstrap Aggregating. Under this Bagging, Random Forest is a famous example of a bagging method that combines multiple decision trees. The concept of decision trees involves training numerous trees on different subsets of data (bootstrap samples) where features and the final prediction are based on a majority vote or averaging[99].

2. Boosting is also known as Adaptive Boosting (AdaBoost). AdaBoost focuses on correcting the

errors made by previous models. Weak learners are sequentially trained, and more weight is given to misclassified instances in each subsequent round[100,101]. Gradient Boosting Trees, XGBoost, LightGBM, and CatBoost build trees sequentially to correct previous mistakes. These are further improvements in gradient boosting techniques with optimized algorithms and better performance.

3. Stacking (also known as Stacked Generalization): Stacking combines the predictions of multiple models by training a meta-learner (a higher-level model) on their outputs[102]. The base models are often diverse regarding algorithms and hyperparameters to reduce bias.

4. Voting: In machine learning, Voting has two methods: Hard Voting and Soft Voting. In Hard Voting, multiple models make predictions where the final decision is based on the majority vote. On the other hand, Soft Voting assigns weights to each model's prediction and averages them instead of counting votes. This method is usually utilized when models provide probability scores[103].

5. Bootstrapped Ensembles: This method, also known as Bootstrap Aggregating (Bagging), involves training multiple models independently on different subsets of the training data and aggregating their predictions to reduce overfitting[104].

6. Random Subspace Method: The Random Subspace Method is similar to bagging but focuses on using different subsets of features for each model.

7. Random Patches: The random Patches method is a combination of bagging and random subspace methods, where models are trained on both different subsets of data and other subsets of features.

8. Bayesian Model Averaging: This approach combines models by averaging their predictions while considering their uncertainties. It uses a Bayesian framework to make more informed predictions.

Ensemble methods can significantly improve model performance, reduce overfitting, and enhance the generalization of machine learning models. The choice of ensemble method depends on the

specific problem, the characteristics of the data, and the computational resources available. It's common to experiment with different ensemble techniques to determine which works best for a particular task.

2.2.3.2 Stack Ensemble Machine Learning

Stacking in machine earning is also known as stacked generalization, is an ensemble machine learning approach that combines the predictions of multiple base models (learners) to create a more robust and accurate final prediction[105, 106]. The idea behind stacking is to leverage the strengths of different models and use their combined wisdom to make better predictions. An overview of the method of stack ensemble machine learning includes

1. Base Models: - Start by selecting a set of diverse base models. These models can be of different types, such as decision trees, random forests, support vector machines, neural networks, k-nearest neighbors, or machine learning algorithms[107]. The diversity among base models is crucial in capturing distinct patterns and reducing overfitting risks.

2. Data Split: - Divide dataset into two or more subsets. The most common approach is to split the data into a training set and a holdout validation set. The training set is used to train the base models, while the validation set is used to make predictions and create a new dataset[108].

3. Base Model Training: - Train each base model on the training data. Each model learns to make predictions based on the features and target variables.

4. Validation : - the validation set is use to obtain predictions from each base model. These predictions serve as inputs to the meta-model.

5. Meta-Model: - Train a meta-model (also known as a blender or aggregator) using the predictions from the base models as features. The meta-model aims to combine the base model predictions and

generate the final prediction[109]. Common choices for the meta-model include linear regression, logistic regression, or even another machine-learning model like a decision tree or random forest.

6. Final Prediction: - Once the meta-model is trained, predictions are make on new, unseen data[110]. The meta-model takes the predictions of the base models as input and produces the final prediction.

7. Cross-Validation: It's a good practice to use cross-validation when training the base models and the meta-model. Cross-Validation helps assess the overall performance and generalization ability of the stacking ensemble.

Stacking can lead to better predictive performance than individual base models[111]. Stacking can handle a wide variety of data types and model classes. Stacking leverages the complementary strengths of different models, making it more robust to outliers and noise in the data[112,113]. Stacking requires more computational resources and time than training a single model. Stacking is a powerful technique for improving the accuracy of machine learning models and is commonly used in data science competitions and real-world applications to achieve better predictive performance.

2.3 Related Studies

Software-Defined Networking (SDN) technology has seen significant development and research. There are numerous related works, research papers, and areas of interest in SDN. Key concepts and associated works in the realm of Software-Defined Networking includes SDN Architecture, Network Virtualization, SDN Controllers, Network Security, Traffic Engineering, Quality of Service (QoS), Multi-Tenant Environments, Wireless SDN,SDN and Internet of Thing (IoT), SDN and 5G, Operational Aspects, Energy-Efficient Networking, Performance Evaluation, SDN Programming Languages and Frameworks.

SDN with respect to Traffic Engineering and Quality of Service (QoS), researchers have attempt to enhance the efficiency and QoS of SDN-based networks. SDN has been used to address critical

challenge in modern networking, where the dynamic and diverse nature of traffic, such as IoT devices, requires more sophisticated approaches to network management and traffic routing. Yusuf et al proposed technique that aims to enhance the efficiency and QoS of SDN-based networks by considering composite metrics and flow classification during path selection[114]. Sayjari et al. study on improving the efficiency and quality of service (QoS) in software-defined wireless sensor networks (SDWSNs) that use the IEEE 802.15.4e Time-slotted Channel Hopping (TSCH) protocol[115]. The results from the study of Sayjari et al. claims to significantly improve network efficiency while preserving QoS levels, as demonstrated through an evaluation that considers various network configurations and application requirements[116].

Guo et al. on Software-Defined Networking (SDN) and a QoS-oriented global multi-path traffic scheduling algorithm called QOGMP algorithm address several common problems in the SDN control layer, such as single path routing, congestion, Quality of Service (QoS) requirements, and high delay[117]. The QOGMP algorithm is novel, improving traffic scheduling in SDN by using deep reinforcement learning-based link weight calculations and multi-path routing to address congestion, QoS requirements, and high delay issues. The success of algorithms depend on various factors, including the specific network environment and the quality of the implementation.

Orozco-Santos et al proposes a solution that leverages SDNs and multi-radio sinks to improve the scalability and performance of Industrial Wireless Sensor Networks, especially in environments where strict control and quality of service are essential, such as in Industry 4.0 applications[118]. This approach aims to increase the number of nodes, packet frequency, and overall network performance while efficiently managing the available spectrum.

Researchers had proposed solution to the problem of network congestion and deteriorating user experience in video streaming by combining Software-Defined Networking (SDN) and Reinforcement Learning (RL)[119]. This research addresses a real-world problem caused by the

exponential growth of video streaming services and provides a potential solution.

Isolani et al used a combined airtime-based network slicing, traffic shaping, user association with Multi-Criteria Decision Analysis (MCDA), and integration of SDN for network control to enhance the performance of IEEE 802.11 with 5G networks in indoor environments, to meet the stringent QoS requirements of mission-critical applications[120,121]. The results indicate improved performance and a reduction in queueing delay, bringing the network closer to meeting MCDA requirements[122].

Njah et al proposed Service and Resource Aware Flow Management (SRAFM) in Software-Defined Smart Digital Campus Networks as an approach to managing network flows in smart digital campus environments. SRAFM combines a programmable architecture, distributed flow characterization, and centralized optimization using MILP. An approximation algorithm is applied to address the complexity of the problem. The evaluation demonstrates significant performance improvements compared to existing benchmarks[123].

i Software-Defined Networking

Software-Defined Networking (SDN) is a revolutionary paradigm in networking that aims to make network infrastructure more flexible, programmable, and responsive to the needs of modern applications and services. SDN is an architectural approach that separates the network's control plane (which decides where traffic should be sent) from the data plane (which forwards the traffic). This separation allows for centralized network management and control, making it easier to manage and adapt network resources. SDN Controller is the centralized control plane that manages and directs traffic in the network. Southbound APIs in SDN is the communication interfaces between the SDN controller and network devices (e.g., switches and routers). Northbound APIs in SDN enable communication between the SDN controller and applications or network services. SDN allows network administrators to adapt the network to specific application needs by reconfiguring

the control plane. It simplifies network management and policy enforcement by centralizing control.

OpenFlow is one of the earliest and most widely adopted SDN protocols, it defines the communication between the controller and network devices[124].

SDN continues to evolve, focusing on improving security and scalability. Integrating SDN with emerging technologies like Edge Computing and IoT is becoming increasingly important. The adoption of SD-WAN (Software-Defined Wide Area Networking) is growing, simplifying the management of wide area networks. Software-Defined Networking has brought a fundamental shift in how networks are managed and is crucial in addressing the evolving demands of modern applications and services. Its adoption has grown steadily, and it will likely remain a central component of networking technologies. However, it's essential to carefully consider its implementation, considering the specific needs and challenges of network environment. DDoS attack detection model integrated with SVM classification algorithms has been used within an SDN environment simulated by Mininet and Floodlight, achieving an average accuracy rate of 95.24%, demonstrating its practical value for network security[125]. KNN and SVM algorithms has been used for a DDoS detection method in SDN, achieving over 99% accuracy with just 11% resource consumption[126].

ii Stack Ensemble Models and SDN

Stacking Ensemble Models and Software-Defined Networking (SDN) are two distinct concepts in the fields of machine learning and computer networking, respectively. A stack ensemble model, also known as stacking or stacked generalization, is a machine learning technique that combines the predictions of multiple base models to create a more robust and accurate predictive model. The idea is to leverage the strengths of various base models to compensate for each other's weaknesses. Ensemble Learning (EL) is a machine learning technique that combines multiple models to improve overall performance[127]. It's often used to reduce the risk of overfitting and enhance the robustness

of the model. Abar et al reported performance metrics, including 99 % accuracy, precision, sensitivity, and specificity, indicate a highly effective model for DDoS prediction. Such a model would be invaluable in enhancing the security of SDN-based networks by rapidly detecting and mitigating DDoS attacks, thus maintaining network availability and reliability[128,129]. Christila and Sivakumar combined Cloud Computing (CC) and Software-Defined Networking (SDN) to enhance the capabilities of cloud networking and introduces a novel approach for detecting and mitigating Distributed Denial of Service (DDoS) attacks in a Cloud-SDN environment using Multi-Layer[130,131]. Houda proposed ensemble learning as a solution to address the limitations of conventional ML/DL-based IDS. to address security concerns and prevent issues such as high variance and bias[132]. Eom et al applied four ensemble algorithms and analyzed their classification performance and found that the LightGBM model achieved the best classification performance[133]. LightGBM is known for its speed and efficiency in handling large datasets, so it's a good choice for real-time network traffic classification[134].

In today's digital landscape, Denial of Service (DOS) or Distributed Denial of Service (DDOS) attacks are a significant threat on the Internet users thereby render critical online resources like servers and bandwidth inaccessible to legitimate users by inundating them with malicious traffic. Shirmarz et al proposed a DDOS attack detection system designed for a Software-Defined Network (SDN) architecture, suitable for deployment within the POX controller and the simulation results demonstrate that the proposed model achieves an impressive accuracy rate of about 99.4%, marking a substantial improvement compared to conventional approaches like Decision Tree (DT), K-Nearest Neighbour (KNN), and Support Vector Machine (SVM)[135].

In response to the challenge of unbalanced data in SDN network streams, Lin and Hongle proposed an adaptive intrusion detection model that enhances traditional block learning. By improving the bagging algorithm with features like unbalanced detection, dynamic penalty factors, and selection integration, the proposed system mitigates the impact of data stream imbalance on classifier

performance. This enhanced online integrated learning algorithm is then applied to SDN network intrusion detection, resulting in improved detection accuracy, particularly in recognizing previously unknown intrusion behaviors, as demonstrated through experiments with the NSL-KDD dataset simulating SDN network data streams[136]. Deepa et al proposed an ensemble technique utilizing various machine learning (ML) algorithms, including K-Nearest Neighbor (KNN), Naive Bayes, Support Vector Machine (SVM), and Self-Organizing Map (SOM), to identify anomalous data traffic behavior within an SDN controller. Experimental results reveal that this ensemble method outperforms individual learning algorithms in terms of accuracy, detection rate, and false alarm rate[137]. Stack ensemble models are a powerful technique for improving predictive accuracy. Still, they need to be used judiciously, and careful consideration be given to the selection of base models and the tuning of hyperparameters to achieve the best results.

2.4 Summary of Gaps in Literature Reviewed

The reviewed literature discusses the gaps in performance of Software-Defined Networking (SDN) and Stack Ensemble Models as follows:

i. Gaps in Reviewed Literature on Software-Defined Networking (SDN)

While SDN has seen significant development, there is a gap in understanding how it can effectively integrate with emerging technologies like Edge Computing and IoT. Further research is needed to explore the seamless integration of SDN with these technologies. SDN is evolving to improve security and scalability. There is a need for more research on enhancing the security of SDN-based networks and addressing scalability issues as networks grow and become more complex. While some research focuses on SDN in industrial environments, more studies are required to explore how SDN can be applied in Industry 4.0 applications and intelligent industrial environments, especially concerning quality of service and scalability[138,139]. SDN solutions for addressing network

congestion in real-time multimedia traffic, such as video streaming, are crucial. More research is needed to optimize SDN for this real-world problem, considering the growing demand for multimedia services. Further research is necessary to investigate how SDN can improve the quality of service and reduce delays for mission-critical applications, for example, in 5G networks. Research gaps exist in optimizing SDN for multi-tenant environments. How can SDN efficiently manage and allocate network resources to multiple tenants while maintaining high performance and quality of service? With increasing environmental concerns, there is a need for research into making SDN more energy-efficient. How can SDN technology be optimized to reduce power consumption and environmental impact? More performance evaluation studies and real-world case studies are needed to assess the practical benefits and challenges of implementing SDN in various network environments. As SDN continues to evolve, research into developing programming languages and frameworks specifically tailored for SDN is essential to simplify network management and application development[140].

ii. Gaps in Reviewed Literature on Stack Ensemble Models

While some studies have explored using ensemble models for intrusion detection and DDoS mitigation in SDN, more research is needed to enhance network security further by applying ensemble techniques. Addressing the challenge of unbalanced data in SDN network streams is essential[141]. Research should continue to develop adaptive techniques within ensemble models to handle this data imbalance effectively. Careful consideration is required when selecting the base models for ensemble techniques. More research should focus on optimizing the selection of base models and tuning hyperparameters to achieve the best results. Further research is required to explore the integration of deep reinforcement learning into ensemble models to enhance traffic scheduling, security, and other SDN-related applications. This can provide valuable insights that could help maintain network availability and reliability. Additionally, real-time detection and mitigation of threats, such as DDoS attacks, using ensemble models in SDN environments is crucial

for ensuring network security. It is crucial to assess continuously and benchmark ensemble models against various threat scenarios to ensure their effectiveness in network security. Although some studies focus on security, the potential applications of ensemble models in SDN are broad. Further research is necessary to explore their possible use in network optimization, quality of service, and traffic management. Additionally, it is crucial to study how these models can adapt to changing network environments and scale to accommodate evolving network requirements. An area for future research is to explore how ensemble models can complement and enhance the management of emerging networking technologies, such as SD-WAN. Upon reviewing the literature, it became apparent that there are still areas where further research and development are needed to advance the understanding and practical implementation of SDN and Stack Ensemble Models in various networking contexts. Addressing these gaps in knowledge can help researchers and practitioners enhance the capabilities and effectiveness of these technologies.

Endnotes

1. S. Dou, G. Miao, Z. Guo, C. Yao, W. Wu and Y. Xia "Matchmaker: Maintaining network programmability for software-defined WANs under multiple controller failures", Computer Networks , Vol. 192 Elsevier p. 108045, 2021

2. A. V. Jha, A. N. Ghazali, B. Appasani, C. Ravariu and A. Srinivasulu. Reliability analysis of smart grid networks incorporating hardware failures and packet loss Rev. Roum. Sci. Tech. El , Vol. 65, No. 3 p. 245-252, 2021
3. N. Todorov, I. Ganchev, M. O'Droma. Internet Performance Profiling of Countries. Advances in Computing and Network Communications: Proceedings of CoCoNet, Volume 2 Springer p. 503-518, 2021
4. Y. Wang and Juan, J. S. Hamiltonicity of the basic WK-recursive pyramid with and without faulty nodes. Theoretical Computer Science , Vol. 562 Elsevier p. 542-556, 2015
5. G. Kumar, N. Dukkipati, K. Jang, H. M. G. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan. Swift: Delay is simple and effective for congestion control in the datacenter. Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication p. 514-528, 2020
6. Y. Lu, G. Zhao, C. Chakraborty, C. Xu, L Yang. Keping Time Sensitive Networking-Driven Deterministic Low-latency Communication for Real-time Telemedicine and e-Health Services. IEEE Transactions on Consumer Electronics, 2023
7. L. Mostacero-Agama and P. Shiguihara. Analysis of Internet Service Latency and its Impact on Internet of Things (IoT) Applications. IEEE Engineering International Research Conference (EIRCON) p. 1-4, 2022
8. P. E. Numan, K. M. Yusof, M. N. B. Marsono, S. K. S. Yusof, M. H. M. Fauzi, S. Nathaniel, E. N. Onwuka and M. A. B. Baharudin. On the latency and jitter evaluation of Software-Defined Networks. Bulletin of Electrical Engineering and Informatics , Vol. 8, No. 4 p. 1507-1516, 2019
9. M. Sahu, S. Damle and A. A. Kherani. End-to-end Uplink Delay Jitter in LTE Systems. Wireless Networks , Vol. 27 Springer p. 1783-1800, 2021
10. R. Aruna, V. S. Kushwah, S. P. Praveen, R. Pradhan, A. J. Chinchawade, R. R. Asaad, R. L. Kumar and R. Lakshmana. Coalescing Novel QoS Routing with Fault Tolerance for Improving QoS Parameters in Wireless Ad-Hoc Network using Craft Protocol. Wireless Networks Springer p. 1-25, 2023
11. V. Arun, M. Alizadeh and H. Balakrishnan. Starvation in End-to-end Congestion Control. Proceedings of the ACM SIGCOMM Conference p. 177-192, 2022
12. A. S. Roque, N. Jazdi, E. P. Freitas and C. E. Pereira. Performance Analysis of In-vehicle Distributed Control Systems Applying a Real-time Jitter Monitor. IEEE 18th International Conference on Industrial Informatics (INDIN) , Vol. 1 p. 663-668, 2020
13. H. Yang, K. Zhan, B. Bao, Q. Yao, J. Zhang and M. Cheriet. Automatic Guarantee Scheme for Intent-driven Network Slicing and Reconfiguration. Journal of Network and Computer Applications , Vol. 190 Elsevier p. 103163, 2021
14. S. Jun, K. Przystupa, M. Beshley, O. Kochan, H. Beshley, M. Klymash, J. Wang, D. Pieniak. A Cost-efficient Software Based Router and Traffic Generator for Simulation and Testing of IP Network. Electronics , Vol. 9, No. 1 MDPI p. 40, 2019
15. A. Chaurasia, S. N. Mishra and S. Chinara. Performance Evaluation of Software-Defined Wireless Networks. IT-SDN and Mininet-WiFi", 2019.
16. H. Habtegebreil, K. Grinnemo, S. Ferlin, P. Hurtig and A. Brunstrom "End-to-end Congestion Control Approaches for High Throughput and Low Delay in 4G/5G Cellular Networks", Computer Networks 186, pp. 107692, 2021
17. S. Chen, S. Sun, G. Xu, X. Su and Y. Cai. Beam-space Multiplexing: Practice, Theory, and Trends, from 4G TD-LTE, 5G, to 6G and Beyond. IEEE Wireless Communications , Vol. 27, No. 2, p. 162-172, 2020
18. Z. Ma, M. Xiao, Y. Xiao, Z. Pang, H. V. Poor and B. Vucetic. High-reliability and Low-latency Wireless Communication for Internet of Things: Challenges, Fundamentals, and Enabling Technologies. IEEE Internet of Things Journal , Vol. 6, No. 5 IEEE p. 7946-7970, 2019

19. W. Guan, H. Zhang and V. C. M. Leung. Analysis of Traffic Performance on Network Slicing using Complex Network Theory. IEEE Transactions on vehicular technology , Vol. 69, No. 12 p. 15188-15199, 2020
20. T. K. Mishra, K. S. Sahoo, M. Bilal, S. C. Shah and M. K. Mishra "Adaptive Congestion Control Mechanism to Enhance TCP Performance in Cooperative IoT", IEEE Access 11, pp. 9000--9013, 2023
21. T. Mazhar, M. A Malik, S. A. H. Mohsan, Y. Li, I. Haq, S. Ghorashi, F. H. Karim and S. M. Mostafa "Quality of Service (QoS) Performance Analysis in a Traffic Engineering Model for Next-Generation Wireless Sensor Networks", Symmetry 15, 2, pp. 513, 2023
22. S. B. Makarov, M. Liu, A. S. Ovsyannikova, S. V. Zavjalov, I. I. Lavrenyuk, W. Xue and J. Qi. Optimizing The Shape of Faster-Than-Nyquist (FTN) Signals with the Constraint on Energy Concentration in the Occupied Frequency Bandwidth. IEEE Access , Vol. 8 IEEE p. 130082-130093, 2020
23. G. Hobbs, R. N. Manchester, A. Dunning, A. Jameson, P. Roberts, D. George, J. A. Green, J. Tuthill, L. Toomey and J. F. Kaczmarek. An Ultra-wide Bandwidth (704 to 4032 MHz) Receiver for the Parket Radio Telescope. Publications of the Astronomical Society of Australia , Vol. 37, p. e012, 2020
24. A. Maghyereh and H. Abdoh. Tail Dependence Between Bitcoin and Financial Assets: Evidence From A Quantile Cross-spectral Approach. International Review of Financial Analysis , Vol. 71 Elsevier p. 101545, 2020
25. Z. Lv, R. Lou, J. Li, A. K. Singh and H. Song. Big Data Analytics for 6G-enabled Massive Internet of Things. IEEE Internet of Things Journal , Vol. 8, No. 7, p. 5350-5359, 2021
26. A. B. Buhendwa, D. Z. Bezgin and N. A. Adams "Consistent and Symmetry Preserving Data-driven Interface Reconstruction for the Level-set Method", Journal of Computational Physics 457, pp. 18. Id/No 111049, 2022
27. H. Zhao, Q. Chen, Y. Qiu, M. Wu, Y. Shen, J. Leng, C. Li and M. Guo "Bandwidth and Locality Aware Task-stealing for Manycore Architectures with Bandwidth-Asymmetric ", Memory , pp. 1â26, 2018
28. R. Damasevicius, A. Venckauskas, S. Grigaliunas, J. Toldinas, N. Morkevicius, T. Aleliunas and P. Smuikys. LITNET-2020: An Annotated Real-world Network Flow Dataset For Network Intrusion Detection. Electronics , Vol. 9, No. 5, p. 800, 2020
29. L. Liu, D. Essam and T. Lynar. On Quantifying the Complexity of IoT Traffic. IEEE 46th Conference on Local Computer Networks (LCN) p. 379-382, 2021
30. B. Yan, Q. Liu, J. Shen, D. Liang, B. Zhao and L. Ouyang. A Survey of Low-latency Transmission Strategies in Software-Defined Networking. Computer Science Review , Vol. 40 Elsevier, p. 100386, 2021
31. R. Boutaba, N.Shahriar, M. A. Salahuddin and N. Limam. Managing Virtualized Networks and Services with Machine Learning. Communication Networks and Service Management in the Era of Artificial Intelligence and Machine Learning. Wiley Online Library p. 33-68, 2021
32. P. Wang, J. Zhang, X. Zhang, Z. Yan, B. G. Evans and W. Wang. Convergence of Satellite and Terrestrial Networks: A Comprehensive Survey. IEEE access , Vol. 8, p. 5550-5588, 2019
33. N. S. Sebopetse, C. R. Burger, M. Mofolo and A. A. Lysko. Measuring with JPerf and PsPing: Throughput and Estimated Packet Delivery Delay Vs TCP Window Size & Parallel Streams. International Conference on Advanced Computing and Communication Systems (ICACCS), 1, 828-832, 2021
34. S. J. Rashid, A. M. Alkababji and A. S. M. Khidhir. Performance Evaluation of Software-Defined Networking Controllers in Wired and Wireless Networks. TELKOMNIKA (Telecommunication Computing Electronics and Control), 21, 49-59, 2023
35. B. Hardin, D. Comer and A. Rastegarnia. On the Unreliability of Network Simulation Results FROM Mininet and iPerf International. Journal of Future Computer and Communication,

12, 2023
36. M. T. Islam, N. Islam and M. A. Refat. Node To Node Performance Evaluation Through RYU SDN Controller. Wireless Personal Communications, Springer, 112, 555-570, 2020
37. H. Iqbal and S. Naaz. Wireshark as a Tool for Detection of Various LAN Attacks. Int. J. Comput. Sci. Eng, 7, 833-837, 2019
38. M. Kotari and N. N. Chiplunkar. Investigation of Security Issues in Distributed System Monitoring Handbook of Computer Networks and Cyber Security: Principles and Paradigms, Springer, 609-634, 2020
39. S. S. Rajawat, P. Khatri and G. Surange. Sniffit: A Packet Sniffing Tool Using Wireshark International Conference on Communication, Networks and Computing, 203-212, 2022
40. S. Bekoe. Network traffic analysis using wireshark in solving network problems, Winneba, 2020
41. L. F.Sikos. Packet Analysis for Network Forensics: A comprehensive Survey. Forensic Science International, 32, 200892, 2020
42. R. Tuli. Packet Sniffing and Sniffing Detection. International Journal of Innovations in Engineering and Technology, 16, 2020
43. L. Perigo, R. Gandotra, D. Gedia, M. Hussain, P. Gupta, S. Bano and V. Kulkarni. Voip security: A Performance and Cost-benefit Analysis. Information Technology in Industry, 8, 2020
44. J. von der Assen. DDoSGrid 2.0: Integrating and Providing Visualizations for the European DdoS Clearing House. University of Zurich, 2021
45. D. S. Rana, S. A. Dhondiyal and S. K. Chamoli . Software-Defined Networking (SDN) Challenges, Issues and Solution. International Journal of Computer Sciences and Engineering,7, 884-889, 2019
46. S. Ahmad and A. H. Mir. Scalability, Consistency, Reliability and Security in SDN Controllers: A Survey of Diverse SDN Controllers Journal of Network and Systems Management, Springer, 29, 1-59, 2021
47. R. Dey and R. Mathur "Ensemble Learning Method Using Stacking with Base Learner, A Comparison", International Conference on Data Analytics and Insights, pp. 159--169, 2023
48. H. Azzawi. Computational Models and Approaches for Lung Cancer Diagnosis. Deakin University, 2019
49. M. M. Mariani and S. F. Wamba. Exploring How Consumer Goods Companies Innovate In The Digital Age: The Role Of Big Data Analytics Companies. Journal of Business Research, Elsevier, 121, 338-352, 2020
50. C. Qin, Y. Zhang, F. Bao, C. Zhang and P. Liu. XGBoost Optimized By Adaptive Particle Swarm Optimization For Credit Scoring Mathematical Problems In Engineering, Hindawi Limited, p 1-18, 2021
51. H. Shamsudin, M. Sabudin and U. K. Yusof "Hybridisation Of RF (Xgb) To Improve The Tree-based Algorithms In Learning Style Prediction", IAES International Journal of Artificial Intelligence 8, 4, pp. 422, 2019
52. D. Wang and X. Yue. The Weighted Multiple Meta-models Stacking Method For Regression Problem. Chinese Control Conference (CCC), 7511-7516, 2019
53. H. Shahabi, B. Jarihani, T. P. Sepideh, D. Chittleborough, M. Avand, O. Ghorbanzadeh. A semi-automated object-based gully networks detection using different machine learning models: a case study of Bowen catchment, Queensland Australia, MDPI Sensors , Vol. 19, No. 22, p. 4893, 201
54. A. Morellos, X. E. Pantazi, D. Moshou, T. Alexandridis, G. Whetton, R.; G. Tziotzios, J. Wiebensohn, R. Bill and A. M. Mouaze. Machine Learning Based Prediction Of Soil Total Nitrogen, Organic Carbon And Moisture Content By Using VIS-NIR Spectroscopy. Biosystems Engineering, 152, 104-116, 2016
55. M. F. Steel. Model Averaging And Its Use In Economics. Journal of Economic Literature,

American Economic Association Nashville, TN 37203-2425, 58, 644-71, 2020
56. M. F. Steel. Model Averaging And Its Use In Economics. Journal of Economic Literature, American Economic Association Nashville, TN 37203-2425, 58, 644-71, 2020
57. M. F. Steel. Model Averaging And Its Use In Economics. Journal of Economic Literature, American Economic Association Nashville, TN 37203-2425, 58, 644-71, 2020
58. A. Abdellatif, H. Mubarak, S. Ahmad, T. Ahmed, G. Shafiullah, A. Hammoudeh, H. Abdellatef, M. Rahman and H. M. Gheni. Forecasting Photovoltaic Power Generation With A Stacking Ensemble Model. Sustainability, MDPI, 14, 11083, 2022
59. J. Tian and J. Zhang "Breast Cancer Diagnosis Using Feature Extraction And Boosted C5.0 Decision Tree Algorithm With Penalty Factor", Mathematical Biosciences and Engineering 19, 3, pp. 2193–2205, 2022
60. R. Dey and R. Mathur. Ensemble Learning Method Using Stacking with Base Learner, A Comparison. International Conference on Data Analytics and Insights, 159-169, 2023
61. M. Hoege, A. Guthke and W. Nowak. The Hydrologist'S Guide To Bayesian Model Selection, Averaging And Combination. Journal of Hydrology, Elsevier, 572, 96-107, 2019
62. M. S. Alkhasawneh. Software defect prediction through neural network and feature selections. Applied Computational Intelligence and Soft Computing, Vol. No. 1 Wiley Online Library p. 2581832, 2022
63. M. W. Nadeem, H. G. Goh, V. Ponnusamy, Y. Aun. DDoS Detection in SDN using Machine Learning Techniques. Computers, Materials & Continua , Vol. 71, No. 1, 2022
64. H. Lu and X. Ma "Hybrid Decision Tree-based Machine Learning Models For Short-term Water Quality Prediction", Chemosphere 249, pp. 126169, 2020
65. H. Lu and X. Ma. Hybrid Decision Tree-based Machine Learning Models For Short-term Water Quality Prediction Chemosphere, Elsevier, 249, 126169
66. Y. Lu, G. Zhao, C. Chakraborty, C. Xu, L. Yang and K. Yu "Time Sensitive Networking-Driven Deterministic Low-latency Communication for Real-time Telemedicine and e-Health Services", IEEE Transactions on Consumer Electronics, 2023
67. M. F. Marino and S. Pandolfi "Hybrid maximum likelihood inference for stochastic block models", Computational Statistics and Data Analysis 171, pp. 19. Id/No 107449, 2022
68. P. Charoenkwan, N. Schaduangrat, C. Nantasenamat, T. Piacham and W. Shoombuatong "iQSP: A Sequence-Based Tool for the Prediction and Analysis of Quorum Sensing Peptides Using Informative Physicochemical", International Journal of Molecular Sciences 21, 1, pp. 75, 2019
69. T. S. Talagala, R. J. Hyndman and G. Athanasopoulos. Meta-learning How To Forecast Time Series. Journal of Forecasting, Wiley Online Library, 42, 1476-1501, 2023
70. M. Frank, D. Drikakis and V. Charissis. Machine-learning Methods For Computational Science And Engineering Computation, MDPI, 8, 15, 2020
71. F. Matloob, T. M. Ghazal, N. Taleb, S. Aftab, M. Ahmad, M. A. Khan, S. Abbas and T. R. Soomro "Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review", IEEE Access 9, pp. 98754--98771, 2021
72. W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. U. Rasool and W. Dou. Complementing IoT Services Through Software-Defined Networking And Edge Computing: A Comprehensive Survey. IEEE Communications Surveys & Tutorials, 22, 1761-1804, 2020
73. N. I. Budi and M. F. Faisal. Web-Based Information Systems: Developing a Design Theory. IOP Publishing Ltd., 2020
74. A. Fraihat "Computer Networking Layers Based On The OSI Model", Test Eng. Manag 83, pp. 6485--6495, 2021
75. M. Mikac and M. Horvati. An Approach For Teaching And Understanding Computer Networks Using Realistic Emulation Tool ICERI2019 Proceedings, 1209-1219, 2019
76. K. Nisar, E. R. Jimson, M. H. A. Hijazi, I. Welch, R. Hassan, A. H. M. Aman, A. H. Sodhro, S. Pirbhulal and S. Khan. A Survey On The Architecture, Application, And Security Of Software-

Defined Networking: Challenges And Open Issues. Internet of Things, Elsevier, 12, 100289, 2020
77. D. Hasan and M. Othman. Efficient Topology Discovery In Software-Defined Networks , Elsevier, 116, 539-547, 2017
78. S. Ahmad and A. H. Mir. Scalability, Consistency, Reliability And Security In SDN Controllers: A Survey Of Diverse SDN Controllers Journal of Network and Systems Management, Springer, 29, 1-59, 2021
79. K. Nisar, E. R. Jimson, M. H. A. Hijazi, I. Welch, R. Hassan, A. H. M. Aman, A. H. Sodhro, S. Pirbhulal and S. Khan. A Survey On The Architecture, Application, And Security Of Software-Defined Networking: Challenges And Open Issues. Internet of Things, Elsevier, 12, 100289, 2020
80. L. J. Horner, K. Tutschku, A. Fumagalli and S. Ramanathan. Virtualizing 5G and Beyond 5G Mobile Network. Artech House, 2023
81.V. Shahrokhkhani. An Analysis on Network Virtualization. Protocols and Technologies, 2021
82.B. K. Mukherjee, S. I. Pappu, M. J. Islam and U. K. Acharjee. An SDN based distributed IoT network with NFV implementation for smart cities Cyber Security and Computer Science. Second EAI International Conference, ICONCS 2020, Dhaka, Bangladesh, February 15-16, Proceedings 2, 539-552, 2020
83. A. Duque-Torres, F. Amezquita-Suárez, O. M. Caicedo-Rendon, A. Ordóñez and W. Y. Campo. An approach based on knowledge-defined networking for identifying heavy-hitter flows in data center networks. Applied Sciences, MDPI, 9, 4808, 2019
84. A. L. Aliyu, A. Aneiba, M. Patwary and P. Bull. A Trust Management Framework For Software-Defined Network (SDN) Controller And Network Applications. Computer Networks, Elsevier, 181, 107421, 2020
85. A. Alghamdi, D. J. Paul and E. J. Sadgrove. A RESTful Northbound Interface for Applications in Software-Defined Networks. WEBIST, 453-459, 2021
86. U. A. Bukar and M. Othman. Architectural Design, Improvement, And Challenges Of Distributed Software-defined Wireless Sensor Networks. Wireless Personal Communications , Vol. 122, No. 3 Springer p. 2395-2439, 2022
87. L. Yang, Liang, N. Bryan, W. K. G. Seah, L. Groves and D. Singh. A Survey On Network Forwarding In Software-Defined Networking. Journal of Network and Computer Applications , Vol. 176 Elsevier p. 102947, 2021
88. M. Priyadarsini and P. Bera "Software-Defined Networking Architecture, Traffic Management, Security, And Placement: A Survey", Computer Networks 192 , pp. 108047, 2021
89. I. H. Sarker. "Machine Learning: Algorithms, Real-world Applications And Research Directions", SN computer science 2, 3 , pp. 160, 2021
90. C. Kaur "Incorporating Sentimental Analysis Into Development Of A Hybrid Classification Model: A Comprehensive Study", International Journal of Health Sciences 6, pp. 1709--1720, 2022
91. J. L. Hardesty "Transitioning From XML To RDF: Considerations For An Effective Move Towards Linked Data And The Semantic Web", Information Technology and Libraries (Online) 35, 1, pp. 51, 2016
92. A. Sagingalieva, A. Kurkin, A. Melnikov, D. Kuhmistrov, M. Perelshtein, A. Melnikov, A. Skolik and D. Von Dollen "Hyperparameter Optimization Of Hybrid Quantum Neural Networks For Car Classification", 2022
93. L. Villalobos-Arias, C. Quesada-López, J. Guevara-Coto, A. Martínez and M. Jenkins "Evaluating Hyper-parameter Tuning Using Random Search In Support Vector Machines For Software Effort Estimation", Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, 2020
94. H. Alibrahim and S. A. Ludwig "Hyperparameter Optimization: Comparing Genetic Algorithm Against Grid Search And Bayesian Optimization", IEEE Congress on Evolutionary Computation (CEC) , pp. 1551--1559, 2021
95. R. C. Rajabu, J. S.Ally and J. F. Banzi "Application of MobileNets Convolutional Neural

Network Model in Detecting Tomato Late Blight Disease", Tanzania Journal of Science 48, 4 , pp. 913--926,2022
96. B. Dudi and V. Rajesh "Optimized threshold-based convolutional neural network for plant leaf classification: a challenge towards untrained data", Journal of Combinatorial Optimization 43, 2 (2022), pp. 312--349.
97. H. M. Gomes, J. Read, A. Bifet, J. P. Barddal and J. Gama "Machine Learning For Streaming 69Data: State Of The Art, Challenges, And Opportunities", ACM SIGKDD Explorations Newsletter 21, 2, pp. 6--22, 2019
98. J. Al-Nabulsi, N. Turab, H. A. Owida, B. Al-Naami, R. De Fazio and P. Visconti, Paolo "IoT Solutions and AI-Based Frameworks for Masked-Face and Face Recognition to Fight the COVID-19 Pandemic", Sensors 23, 16 , pp. 7193, 2023
99. I. D. Mienye and Y. Sun "A survey of ensemble learning: Concepts, algorithms, applications, and prospects", IEEE Access 10 , pp. 99129--99149,2022
100. X. Tao, Q. Li, W. Guo, C. Ren, C. Li, R. Liu and J. Zou "Self-adaptive Cost Weights-based Support Vector Machine Cost-sensitive Ensemble For Imbalanced Data Classification", Information Sciences 487 , pp. 31--56, 2019
101. A. Zabian and A. Z. Ibrahim "Hybrid Mathematical Model For Data Classification And Prediction: Case Study COVID-19", International Journal of Mathematics and Computer Science 17, 3 (2022), pp. 995--1006.
102. S. Zian, S. A. Kareem and K. D. Varathan "An Empirical Evaluation Of Stacked Ensembles With Different Meta-learners In Imbalanced Classification", IEEE Access 9 , pp. 87434--87452, 2021
103. N. H. Daneshvar, Y. Masoudi-Sobhanzadeh and Y. Omidi "A Voting-based Machine Learning Approach For Classifying Biological And Clinical Datasets", BMC bioinformatics 24, 1 , pp. 1--17, 2023
104. Z. P. Brodeur, J. D. Herman and S. Steinschneider "Bootstrap Aggregation And Cross-validation Methods To Reduce Overfitting In Reservoir Control Policy Search", Water Resources Research 56, 8 , pp. e2020WR027184, 2020
105. O. D. Okey, S. S. Maidin, P. Adasme, R. R. Lopes, M. Saadi, M. D. Carrillo and R. D. Zegarra "BoostedEnML: Efficient Technique For Detecting Cyberattacks In IoT Systems Using Boosted Ensemble Machine Learning", Sensors 22, 19 , pp. 7409, 2022
106. A. Peimankar, T. S. Winther, A. Ebrahimi and U. K. Wiil "A Machine Learning Approach For Walking Classification In Elderly People With Gait Disorders", Sensors 23, 2 , pp. 679, 2023
107. M. Bansal, A. Goyal and A. Choudhary "A Comparative Analysis Of K-nearest Neighbor, Genetic, Support Vector Machine, Decision Tree, And Long Short Term Memory Algori.", Decision Analytics Journal 3 , pp. 100071, 2022
108. R. Shwartz-Ziv and A. Armon "Tabular Data: Deep Learning Is Not All You Need", Information Fusion 81 , pp. 84--90, 2022
109. Y. Cheng, M. Yu, X. Guo and B. Zhou "Few-shot learning with meta metric learners", arXiv preprint arXiv:1901.09890, 2019
110. T. Hartmann, A. Moawad, C. Schockaertc, F. Fouquet and T. Y. Le "Meta-modelling Meta-learning", International Conference on Model Driven Engineering Languages and Systems (MODELS) , pp. 300--305, 2019
111. V. Coscrato, I. M. H. Almeida and R. Izbicki "The NN-Stacking: Feature Weighted Linear Stacking Through Neural Networks", Neurocomputing 399, pp. 141--152, 2020
112. N. Schetakis, D. Aghamalyan, P. Griffin and M. Boguslavsky "Review Of Some Existing QML Frameworks And Novel Hybrid Classical–Quantum Neural Networks Realising Binary Classification For The Noisy Datasets", Sci. Rep. 12, 1, pp. 11927, 2022
113. Y. Trochun, S. Stirenko, O. Rokovyi, O. Alienin, E. Pavlov and Y. Gordienko "Hybrid Classic-Quantum Neural Networks for Image Classification", 2021

114. M. N. Yusuf, K. bin Abu Bakar, B. Isyaku, A. H. Osman, M. Nasser and F. Elhaj "Adaptive Path Selection Algorithm with Flow Classification for Software-Defined Networks", Mathematics 11, 1404 , pp. 1404, 2023
115. T. Sayjari, R. M. Silveira and C. B. Margi "Application-Aware Scheduling for IEEE 802.15.4e Time-Slotted Channel Hopping Using Software-Defined Wireless Sensor Network", Sensors 23, 7143, pp. 7143, 2023
116. T. Sayjari, R. M. Silveira and C. B. Margi "Application-Aware Scheduling for IEEE 802.15.4e Time-Slotted Channel Hopping Using Software-Defined Wireless Sensor Network", Sensors 23, 7143, pp. 7143, 2023
117. Y. Guo, G. Hu and D. Shao "QOGMP: QoS-oriented Global Multi-path Traffic Scheduling Algorithm In Software-Defined Network", Scientific Reports 12, 1 , pp. 1--12, 2022
118. F. Orozco-Santos, V. Sempere-Paya, J. Silvestre-Blanes and J. Vera-Perez "Scalability Enhancement on Software-defined Industrial Wireless Sensor Networks ", IEEE Access 10, pp. 107137–107151, 2022
119. M. A. Jameel, T. Kanakis, S. Turner, A. Al-Sherbaz and W. S. Bhaya "A Reinforcement Learning-Based Routing for Real-Time Multimedia Traffic Transmission over Software-Defined Networking", Electronics 11, 2441 , pp. 2441, 2022
120. P. H. Isolani, D. J. Kulenkamp, J. M. Marquez-Barja, L. Z. Granville, S. Latre and V. R. Syrotiuk "Support for 5G Mission-Critical Applications in Software-Defined IEEE 802.11 Networks", Sensors 21, 693 , pp. 693, 2021
121. W. Guan, H. Zhang and V. C. M. Leung "Analysis of traffic performance on network slicing using complex network theory", IEEE Transactions on vehicular technology 69, 12, pp. 15188--15199, 2020
122. P. H. Isolani, D. J. Kulenkamp, J. M. Marquez-Barja, L. Z. Granville, S. Latre and V. R. Syrotiuk "Support for 5G Mission-Critical Applications in Software-Defined IEEE 802.11 Networks", Sensors 21, 693 , pp. 693, 2021
123. Y. Njah, C. Pham and M. Cheriet "Service and Resource Aware Flow Management Scheme for an SDN-Based Smart Digital Campus Environment", IEEE Access 8 , pp. 119635–119653, 2020
124. G. Yang, H. Jin, M. Kang, G. J. Moon and C. Yoo "Network Monitoring For SDN Virtual Networks", IEEE Conference on Computer Communications, pp. 1261--1270, 2020
125. J. Ye, X. Cheng, J. Zhu, L. Feng, L. Song. A DDoS Attack Detection Method Based on SVM
in Software Defined Network. Security and Communication Networks, Vol. 2018, No. 1 Wiley Online Library p. 9804061, 2018
126. Z. Ma, B. Li. A DDoS attack detection method based on SVM and K-nearest neighbour in SDN environment. International Journal of Computational Science and Engineering , Vol. 23, No. 3 Inder-science Publishers (IEL) p. 224-234, 2020
127. F. Amarudin and W. Ridi "New Approach of Ensemble Method to Improve Performance of IDS using S-SDN Classifier", Solo, Indonesia IEEE, pp. 463--468, 2022
128. T. Abar, A. B. Letaifa and S. E. Asmi "Real Time Anomaly detection-Based QoE Feature selection and Ensemble Learning for HTTP Video Services", Hammamet, Tunisia IEEE, pp. 1--6,2019, 2019
129. A. Rai, P. D. Vyavahare and A. Jain "Distributed Dos Attack Detection And Mitigation In Software Defined Network (SDN)", Proceedings of Recent Advances in Interdisciplinary Trends in Engineering & Applications (RAITEA), 2019
130. S. A. Christila and R. Sivakumar "Multi-Layer Ensemble Deep Reinforcement Learning Based DDoS Attack Detection And Mitigation In Cloud-SDN Environment", Bangalore, India IEEE, pp. 451–455, 2022
131. R. U. Rasool, U. Ashraf, K. Ahmed, H. Wang, W. Rafique and Z. Anwar "Cyberpulse: A

Machine Learning Based Link Flooding Attack Mitigation System for Software Defined Networks", IEEE Access 7, pp. 34885--34899, 2019
132. Z. A. Houda, B. Brik and L. Khoukhi "Ensemble Learning for Intrusion Detection in SDN-Based Zero Touch Smart Grid Systems", Edmonton, AB, Canada: IEEE, pp. 149--156, 2022
133. W. Eom, Y. Song, C. Park, J. Kim, G, Kim and Y. Cho "Network Traffic Classification Using
Ensemble Learning in Software-Defined Networks", Jeju Island, Korea (South) IEEE, pp. 089--092, 2021
134. T. Zou, Y. Wang and W. U. Chengrong "Review Of Network Background Traffic Classification And Identification", Journal of Computer Applications 39, 3, pp. 802, 2019
135. A. Shirmarz, A. Ghaffari, R. Mohammadi and S, Akleylek "DDOS Attack Detection Accuracy Improvement in Software-Defined Network (SDN) Using Ensemble Classification", Ankara, Turkey: IEEE, pp. 111--115, 2021
136. Z. Lin and D. Hongle "Research On SDN Intrusion Detection Based On Online Ensemble Learning Algorithm", Haikou City, China: IEEE, pp. 114--118, 2020
137. V. Deepa, K. M. Sudar and P. Deepalakshmi "Design of Ensemble Learning Methods for DdoS Detection in SDN Environment", Vellore, India: IEEE, pp. 1--6, 2019
138. K. Raza “Computational Intelligence In Oncology. Applications In Diagnosis, Prognosis And Therapeutics Of Cancers” Singapore: Springer, vol. 1016, 2022
139. M. E. Omeka, O. Igwe, O. S. Onwuka, O. M. Nwodo, S. I. Ugar, P. A. Undiandeye and I. E. Anyanwu "Efficacy Of GIS-based AHP And Data-driven Intelligent Machine Learning Algorithms For Irrigation Water Quality Prediction In An Agricultural-mine District Within The Lower Benue Trough, Nigeria", Environmental Science and Pollution Research, pp. 1--30, 2023
140. F. Estrada-Solano, A. Ordonez, L. Z. Granville and O. M. C. Rendon "A Framework For SDN Integrated Management Based On A CIM Model And A Vertical Management Plane", Computer Communications 102, pp. 150--164, 2017
141. F. Grina, Z. Elouedi and E. Lefèvre "Uncertainty-aware Resampling Method For Imbalanced Classification Using Evidence Theory" Cham: Springer, pp. 342--353, 2021

Chapter Three
Methodology

3.1 Research Approach

This research used a quantitative approach to carry out computer network experiments in order to investigate factors identified for the study of the performance of combined machine learning models (KNN, SVM_RBF, DT, RF, and MLP) on the prediction of the influence of Background-Traffic and Bandwidth-Limit on computer networks. Three experiments were carried out in this study. The first experiment investigates the effects of traditional and Software-Defined Networking on the

performance parameters of a computer network. The second experiment observed the influence of Background-Traffic and Bandwidth-Limit on SDN, while the third experiment investigated the performance of combined machine learning on the dataset obtained in the second experiment. The Command Line Interface (CLI) in bash and Graphic User Interface (GUI) in miniEdit were used to interact with the emulation system (Mininet), while Jupyter Notebook was used to interact with Jupyter kernels for machine learning work. A jupyter kernel is a runtime environment that provides programming support for the Jupyter Notebook apps.

3.2 Research Design

The performance of networks (using the Mininet emulator and physical personal computers) was measured and reported in this study. Laboratory experiments were deployed for this research using physical and emulated networks with end devices and intermediary devices to investigate quality of service (QoS) for the computer network performance and performance of combined machine learning. The architectural design for this research consists of the SDN controller plane, data plane, and application plane, as shown in Figure 3.1. Experiments for research objectives one and two used factorial experiments designed to know the effect of interaction between and among the levels of each factor deployed for the experiments (see Section 1.3). On the other hand, experiments for objective three involve the application of datasets obtained in objective 2. These datasets are subsequently analyzed using stacked machine learning models to forecast the performance of software-defined networking (SDN) in the presence of background traffic and bandwidth constraints. This approach allows for a detailed examination of how SDN functions under specific network conditions, providing valuable insights into its performance and capabilities.

Figure 3.1: Architectural design for TCN and SDN Experiment

Source: Author's concept (2021)

The experiments for this research were emulated using Mininet running on Ubuntu 22.04 LTS operating system on Lenovo Thinkpad T420 with 8 GB RAM, Intel Core i7 CPU, 2.7 GHz. Physical computers and virtual personal computers were used to set up computer networks as traditional computer networks and software-defined networks (SDN). Mininet, an emulation software, was used to deploy computer networks. In the Mininet, legacy switches were adopted for traditional computer networks, while OpenFlow switches were used for SDN. The networks include end devices (personal computers) and intermediary devices, i.e., switches, routers, and POX controllers. Two physical Wi-Fi routers (Bolt Huawei E5573CS-322 Zong Bolt+) were used to

route traffic from the virtual and physical PCs in the Local Area Networks (LAN) to the Internet. The OpenFlow switches were attached to a Wi-Fi router (Bolt Huawei E5573CS-322 Zong Bolt+) to connect the LANs to Internet through telco. Custom network typologies were written for experiments one and two using Python version 3.8.8. Each topology was executed using Mininet version 2.3.0 emulation to test the computer network performance. Anaconda version 3 was installed to use Python libraries for machine learning and run Jupyter Notebook for machine learning activities on the local machine used in this study.

3.3 Research Method

Three experiments were carried out in this study to achieve the set objectives, data obtained were analyzed, and conclusions were drawn on the performance of the computer network.

3.3.1 Effect of Traditional Computer Networking (TCN), Software-defined Networking (SDN) and Hybrid TCN with SDN on Performance of Computer Network

In order to investigate the effect of traditional and software-defined networking on performance parameters of computer network, fifty hosts from each switch (s1, s2, s3, s4, and s5) were attached to different computer networking - traditional computer networking (D0), software-defined networking (D1), and combined traditional with software-defined network (D2) emulated using default Mininet emulation tool version 2.3.0. The network topology for the investigation was based on the research architectural design, as shown in Figure 3.1. Figures 3.2, 3.3, and 3.4 show the network topology for traditional computer networking (D0), software-defined networking (D1), and combined traditional with software-defined network (D2), respectively. Each of the connected links had bandwidth allocation (10 Mbps) following the procedure of Chaudhari et al[1].

Figure 3.2: Traditional computer networking (D0)

Source: Author's Network Topology (2021)

Figure 3.3: Software-Defined Networking (D1)

Source: Author's Network Topology (2021)

Figure 3.4: Combined traditional with Software-Defined Network (D2)

Source: Author's Network Topology (2021)

The three factors of a factorial experiment were set up in a completely randomized design (D3A5P5) with ten replicates each on the performance of the computer network based on traditional and Software-Defined Networking. The network type (N0, N1, and N2) also represented as (D0, D1, and D2) is the first factor. The second factor is the Packet Internet groper (Ping) command representing a running application on a host under s1 that sends five (5) different levels (100, 90, 80, 70 % of 65,507 bytes, and 10,000 bytes) engaging the server's services. According to IETF RFC 793 and 768, the theoretical maximum size for Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) is about 64 kB (TCP 65535 bytes and UDP 65507 bytes)[2]. However, packets of this size are not commonly used due to the Maximum Transmission Unit (MTU), the maximum data size that can be sent at a time. This means that data with a size larger than the MTU is passed from the transport layer to the network layer, and the data are fragmented into smaller packages that are provided with the IP header with the final destination address. The third factor is the number of packets (32, 64, 128, 256, and 512) sent across the network. To generate traffic, the

Iperf command was executed using '*sudo iperf -s -p 5566 -i 1*' to set up the Iperf server for TCP on a server's host Command-Line Interface (CLI) on s5, and '*sudo iperf -s -u -p 5566 -i 1*', for setting Iperf server for UDP traffic on server's host CLI on s5. Iperf client connection from PC3 and PC4 were connected to the TCP and UDP server using '*sudo iperf -c <server address> -p 5566 -i 1*' and '*sudo iperf -c <server address> -u -b 10M –t 15 -p 5566*' respectively. The experiment was carried out using the Mininet emulation tool. Mininet runs the collection of end devices (i.e., the PCs), intermediary devices (i.e., switches and routers), and links (i.e., the media connecting the end devices and intermediary devices) on a single Linux kernel. The results from computer network experiment one using Mininet emulation were recorded. A cleanup process for the experiment was run at the end of every new network topology execution to avoid any previous logs and cache data building up with the current experiment[3].

3.3.2 Effect of Background-Traffic and Bandwidth-Limit on the performance (latency, bandwidth, throughput, jitter and datagram loss) of Software-defined networking (SDN)

In this section, virtual and physical PCs were connected using Mininet. The Mininet was also connected to the Internet to investigate the effect of Background-Traffic and Bandwidth-Limit on the performance (latency, bandwidth, throughput, jitter, and datagram loss) of software-defined networking (SDN) Mininet was installed as an emulation system on PC1, tagged as 'Mininet PC'. The 'Mininet PC' was used to set up the connection of virtual PCs as the end devices and intermediary devices like switches POX SDN controller using Python code. The Mininet and POX controller were initialized through the command line interface. POX controller was remotely connected to the Mininet using '*sudo ./pox.py forwarding.l2_learning*'. The software-defined network was set up on the 'Mininet PC' consisting of a controller and nine openvswitch with 28 PCs provisions on each switch (Figure 3.5). In the Python code, the open switches were looped to form a stacked switch (Appendix A). In the SDN looped up switches, one edge switch was bridged with the first access point or wireless router (WR1) that connected the physical PCs, while the other edge switch was bridged with another wireless router (WR2) that connected the virtual PCs and

some other physical PCs to the Internet (Figure 3.6 and Appendix A). After the successful connectivity test, the network experiments were set up for Background-Traffic and bandwidth tests to measure QoS (throughput, bandwidth, latency, and jitter) for computer network performance.

Figure 3.5: Mininet Topology Design

Source: Author's Network Topology Design(2021)

Figure 3.6: Architectural design for Background-Traffic and Bandwidth-Limit Experiment

Source: Author's Network Architectural design (2021)

3.3.2.1 The Mininet PC

The Mininet PC was set up by performing a clean installation of the Ubuntu 22.04 LTS operating system on the Lenovo Thinkpad. The Linux machine was updated to ensure that applications with their dependency files on the Linux system were updated. Custom network typology was written

using Python version 3.8.8. The topology was implemented using Mininet version 2.3.0 emulation to set up a virtual computer network and physical LAN for the SDN environment. Mininet was installed based on the native installation from source[4]. The virtual computer network was attached to two wireless routers (Bolt Huawei E5573CS-322 Zong Bolt+) to connect physical PCs with the SDN and connect all the end devices to the Internet. All the virtual and physical PCs on the network were set on 192.168.8.0/24 network address for this study with a provision of the first useable IP address assigned to WR1 as the gateway, the last useable IP address was assigned to WR2 and 28 IP addresses were assigned to each set of 28 PCs on s1 to s9. All traffic to the Internet was routed via WR1 with 192.168.8.1/24. The DNS address (8.8.8.8) was configured on PCs participating in the Internet connection. A network reachability test was carried out to ensure the LAN PCs were reachable to one another. In order to route traffic from the wireless router, the installed openvswitch on the Mininet s1 and s9 were bridged with the wireless router using the command *sudo ovs-vsctl add-port s1 enx0c5b8f279a64* and *sudo ovs-vsctl add-port s9 eth0* respectively. Where *enx0c5b8f279a64* and *eth0* are the ports on the PC1 acting as the openvswitch, i.e., the Mininet PC.

3.3.2.2 Data collection

Bash scripts were created to execute the necessary command and save the results in separate files for each experimental run. In the Bandwidth-Limit experiment, the iperf command was executed in both server and client modes while allowing different levels of bandwidth on the network between the server and client machines. Additionally, various Background-Traffic was running on the network simultaneously. The PC with IP address 192.168.8.228 was utilized as the Iperf server.

i. Bandwidth-Limit (U**) test**

A script '*iperf -u -s -p 5566 -i 33.3 > > UDP.txt*' for the iperf udp test was run for 100 sec for each test, and the results were saved in UDP1.txt. The command for Bandwidth-Limit (U) test

running Iperf for UDP in client mode was also run for 'no Bandwidth-Limit (U1)', '256M Bandwidth-Limit (U2), and 512M Bandwidth-Limit (U3) respectively as shown Listing 3.1.

Listing 3.1: Iperf for UDP in client mode for Bandwidth-Limit test

```
iperf -c address -u -p 5566 -t 100 >> UDP_U1_01.txt

iperf -c address -u -b 256M -p 5566 -t 100 >> UDP_U2_01.txt

iperf -c address -u -b 512M -p 5566 -t 100 >> UDP_U3_01.txt
```

ii. TCP test

TCP test was equally carried out using the command '*iperf -s -p 5566 -i 100 > > TCP.txt*' along the tests during the UDP tests as shown in Figure 3.7.

iii. Background-Traffic (T) tests

Background-Traffic (T) tests were conducted using default 64 bytes (T1) in Linux, 58956 bytes (T2), and 65507 bytes (T3) as traffic to flood the gateway node as well as non-stop engaging DNS server (8.8.8.8) with default ping using ping command from another host (see Listing 3.2). While Background-Traffic tests were taking place, some PCs were also made to use various cyberservices - email agent using Thunderbird, web browser, and wget index page of 'Leadcity University Ibadan' website (see Figure 3.8). The following commands were used for Background-Traffic test (T1, T2, and T3)

Listing 3.2: Background-Traffic test

```
ping 8.8.8.8 -c 100 -i 1 >> T1.txt

ping 192.168.8.1 -c 100 -i 1 -s 58956 >> T2.txt
```

```
ping 192.168.8.1 -c 100 -i 1 -s 65507 >> T2.txt
```

Figure 3.7: Running bash scrip for data collection on the influence of backgroung traffic and Bandwidth-Limit

Source: Author's Network Observation (2022)

Figure 3.8: Access various cyber services

Source: Author's Network Observation (2022)

Treatment T1 used a virtual PC (host 50) to ping a Google DNS server with 100 packets, T2 ping LAN gateway 192.168.1 with byte size -s 58956 using a physical PC while wget the index page of LCU website www.lcu.edu.ng. T3 used a virtual PC on Mininet using xterm h60 to flood the LAN gateway using 'ping 192.168.1 -s 65507.'

iv. Network flow

Wireshark and Open vSwitch were used to capture data traffic in this study. For the Wireshark, the tshark command was used to capture traffic from the gateway interface 'enx0c5b8f279a64' in the topology for this network experiment. The following command was used in bash to capture data flow and save it in a nlog.pcap file.

tshark -w nlog.pcap -i enx0c5b8f279a64 -c 100

In OVS, flows are automatically managed by the SDN controller. POX controller is used in this study, while the dump-flows command was modified using a bash script to copy all flow entries in the switch's tables that match flows to a file.

ovs-ofctl dump-flows s$i > > s$i.csv

All the flows from the switches in the topology were captured and processed for feature classification using machine learning algorithms (Figure 3.9).

Figure 3.9: Data captured on the influence of Background-Traffic and Bandwidth-Limit

Source: Author's Network data collection (2022)

3.3.3 Evaluation of combined machine learning model on the influence of Background-Traffic and Bandwidth-Limit on the performance of software-defined networking (SDN)

The dataset obtained in section 3.3.2 was explored to view the dataset features, measure of central tendency and variability of the dataset, correlation, and dataflow features from the dataset on the influence of Background-Traffic and Bandwidth-Limit on the network on the performance of software-defined networking as shown in Figure 3.10 and Appendix B.

Figure 3.10: Stack machine learning models flowchart

Source: Author's flowchart (2022)

Stacking machine learning models were used to combining multiple individual models to improve predictive performance by leveraging their strengths and mitigating their weaknesses. In this approach, base models: K-Nearest Neighbors (KNN), Support Vector Machine with Radial Basis Function Kernel (SVM_RBF), Decision Trees (DT), Random Forest (RF), and Multi-Layer

Perceptron (MLP) were trained separately on the same dataset. KNN offers simplicity and effectiveness in local data structures, while SVM_RBF provides robustness in high-dimensional spaces. DTs contribute interpretability, and RFs add ensemble strength and reduce overfitting. MLPs introduce deep learning capabilities for capturing complex patterns. The outputs of these base models were then fed into a meta-model, which typically employs a simpler algorithm, to learn how to best combine their predictions. This stacked model aims to achieve superior accuracy and generalization compared to any single model, benefiting from the diverse methodologies and perspectives of the individual algorithms. The performance of stacked or combined models (StkMdl) using machine learning models KNN+SVM_RBF (denoted as 2-StkMdl), KNN+SVM_RBF+DT (denoted as 3-StkMdl), and KNN+SVM_RBF+DT+RF+MLP (denote as 5-StkMdl) with 35 % dataset for the test was evaluated in order to determine the percentage accuracy score (Accuracy), Matthews correlation co-efficiency (MCC) and F1 score (see Appendix C to E). Train/test observation (to validate the models) were set 65/35, 75/25, and 85/15 in order to evaluate the best train/test level on different machine learning models on the generated dataset. The accuracy of these models was observed and recorded.

3.3.3.1 Data collection

Data related to the quality of service, including bandwidth, throughput, latency, jitter (variance in latency), and percentage of datagram loss due to errors, were recorded and analyzed. Ping commands from a computer host for an IPv4 packet with a size range from 1 - 65,507 bytes were used to send Internet Control Message Protocol (ICMP) echo requests/replies to test the connectivity and reachability as well as to flood targeted computer nodes on networks. Bandwidth-Limit was carried out using Iperf UDP bandwidth of 1 - 512 M on the network links. The data collected during the experiment was recorded for subsequent analysis and reporting, utilizing the R Commander and Scikit-learn platforms. Scikit-learn, a widely utilized machine learning library in

Python, presents a comprehensive and user-friendly solution for various machine learning applications, encompassing classification, regression, clustering, and dimensionality reduction[5]. The seamless interface provided by NumPy, SciPy, Matplotlib, and Scikit-learn supports a diverse array of machine learning algorithms [6,7,8].

3.4 Research Test-bed

The primary objectives of this research encompassed a comprehensive exploration of the performance characteristics of various networking paradigms, namely traditional computer networking (TCN), software-defined networking (SDN), and the hybrid integration of TCN-SDN. The investigation delved into key metrics such as latency, bandwidth, throughput, and jitter, aiming to provide insights into their impact on packet flow and the execution of applications within a computer network.

In addition to evaluating the fundamental networking architectures, the research also sought to elucidate the influence of background traffic and bandwidth limitations on the performance of SDN. This involved a nuanced examination of latency, bandwidth, throughput, jitter, and datagram loss within the SDN framework, under different scenarios of network congestion and resource constraints.

Furthermore, the study delved into the intersection of machine learning and networking, specifically focusing on the performance of combined machine learning models. The research aimed to discern how these models could effectively mitigate the effects of background traffic and bandwidth limitations on SDN performance. By employing machine learning techniques, the study sought to uncover potential strategies for optimizing SDN in the face of dynamic and challenging network conditions.

In summary, the research aimed to contribute valuable insights into the comparative performance of

TCN, SDN, and hybrid TCN-SDN. Additionally, it delved into the nuanced impact of background traffic and bandwidth constraints on SDN, exploring the potential of machine learning models to enhance the adaptive capabilities of SDN in complex networking environments. The multifaceted nature of these objectives underscores the research's commitment to providing a holistic understanding of networking performance and optimization strategies. This research was conducted and evaluated at the Computer Network Laboratory in the Computer Science Department, Gateway (ICT) Polytechnic, Saapade, Ogun State.

3.5 Data Analysis, Testing and Evaluation

The data obtained from the experiment was analyzed using the statistical procedures of R Commander and Jupyter Notebook with the Scikit-learn stack on Anaconda3 running on a local PC. The following steps were performed:

1. Analysis of Variance (ANOVA) was used to determine if there were any statistically significant differences between the means of independent groups.

2. Post-ANOVA, New Duncan Multiple-Range Tests were conducted to compare the means of different groups. These tests helped to identify specific differences between group means. The statistical analysis was performed at a significance level of 0.05 to address the stated research questions.

Endnotes

1. J. P. Chaudhari, D. K. Kirange, K. S. Bhagat and S. D Patil . Ealuation of Bandwidth

Utilization in SDN. Journal of Xi'an Shiyou University ISSN No , Vol. 1673 p. 064X, 2019
2. L. L. Zulu, K. A. Ogudo and P. O. Umenne. Emulating Software-defined Network Using Mininet And OpenDaylight Controller Hosted On Amazon Web Services Cloud Platform To Demonstrate A Realistic Programmable Network. International Conference on Intelligent and Innovative Computing Applications (ICONIC), 1-7, 2018
3. I. B. Sadiku, W. Ajayi, W. Sakpere, T. John-Dewole and R. A. Badru. Effect Of Traditional And Software-Defined Networking On Performance Of Computer Network. Scientific Journal of Informatics 9, 2, pp. 111--122, 2022
4. J. Bhatia and P. Prakash. Mininet--An Emulator for Prototyping Large Network Topologies on a Single Machine. Admin Insight, pp. 51--56, 2015
5. E. Côme, N. Jouvin, P. Latouche and C. Bouveyron. Hierarchical Clustering With Discrete Latent Variable Models And The Integrated Classification Likelihood. Advances in Data Analysis and Classification (ADAC) 15, 4, pp. 957--986, 2021
6. G. J. Babu, D. Banks, H. Cho, D. Han, H. Sang and S. Wang. A Statistician Teaches Deep Learning. Journal of Statistical Theory and Practice 15, pp. 1--23, 2021
7. Z. Zhou. Ensemble Learning. Springer Singapore, 181--210, 2021
8. A. R. Mohammed, S. A. Mohammed and S. Shirmohammadi. "Machine Learning and Deep Learning Based Traffic Classification and Prediction in Software Defined." International Symposium on Measurements & Networking (M&N). IEEE, 2019

Chapter Four
Results and Discussion

4.1 Overview

This chapter provides information concerning the results and discussion of the data obtained in the

cause of the experiments stated in previous sections to achieve the stated aim for this research. Also, the findings from the data analysis, the implication of the results concerning the research questions, as well as results from other researchers are presented in this chapter. This section provide response to the research questions raised that: how should the levels of bandwidth limitations for applications running on a computer network be determined in a situation where the network performance starts to degrade noticeably; why is the influence of background traffic and bandwidth limitations on the performance of Software-Defined Network (SDN) significant; how could machine learning models accurately predict network performance under varying conditions of background traffic and bandwidth limitations for optimized network performance in real time? The study was conducted in a computer network lab by connecting LANs to the internet in both emulation and physical modes[1]. The data obtained were analyzed using R commander to provide information on the influence of computer networks on applications running on computers in computer networks[2]. Also, the dataset obtained from the packet flow on SDN was analyzed using sklearn modules in anaconda3 to evaluate the performance of combined machine-learning models.

4.2 Effect of Influence (latency, bandwidth, throughput, and jitter) of Traditional Computer Networking (TCN), Software-Defined Network (SDN), and Hybrid TCN - SDN on Packet Flow and Application Running on a Computer in Computer Network

This section provides the results and discussion of the investigation of the performance (latency, bandwidth, throughput, and jitter) of Traditional Computer Networking (TCN), Software-Defined Network (SDN), and hybrid TCN - SDN on packet flow and application running on a computer in a computer network. The section provide response to the research question raised that how should the levels of bandwidth limitations for applications running on a computer network be determined in a situation where the network performance starts to degrade noticeably? The study assessed the performance impact (in terms of bandwidth, throughput, latency, and jitter) across Traditional Computer Networking (TCN), Software-Defined Network (SDN), and hybrid TCN-SDN systems

on packet flow and application performance. Results indicated that lower packet counts (e.g., 32 packets) led to higher bandwidth usage, as observed in a 65,507 bytes application running on a TCN network, which utilized 10.62 Gbps of bandwidth. In contrast, a 10,000-byte application in a hybrid network used only 8.46 Gbps with 258 packets. These findings suggest that controlling the packet flow and managing the application size are crucial in optimizing bandwidth usage. Additionally, the study found that lower packet counts also correlated with higher throughput, as evidenced by the 18.52 GB throughput in the same TCN network setup with fewer packets. The analysis highlighted that hybrid networks, while offering flexibility, might introduce additional latency and jitter, potentially impacting real-time application performance. Thus, in situations where network performance degradation is detected, adjusting the bandwidth allocation based on the packet flow, application size, and network type can help maintain optimal performance levels. The use of integrated machine learning models can further refine these adjustments by predicting and responding to traffic patterns and network conditions in real time.

4.2.1 Effect of the performance (of the bandwidth, throughput, latency, and jitter) of Traditional Computer Networking (TCN), Software-Defined Network (SDN), and hybrid TCN-SDN on packet flow and application running on a computer in a computer network

Results of the effect of mean of the performance of the bandwidth of Traditional Computer Networking (TCN), Software-Defined Network (SDN), and hybrid TCN - SDN on packet flow and application running on a computer in a computer network are presented in Figure 4.1. The results showed that the 10,000-byte application running in a combined traditional computer network and software-defined network with 258 packets sent across the network, i.e. A5N2P4, had the most minor bandwidth usage of 8.46 Gbps. Also, the 65,507 bytes application running in traditional computer network with 32 packets sent across the network, i.e. A1N0P1, had the highest bandwidth usage of 10.62 Gbps. This indicates that the fewer packets sent across the network, the higher the bandwidth utilized.

Figure 4.2 shows the results of the effect of the mean of performance (throughput) of Traditional Computer Networking (TCN), Software-Defined Network (SDN), and hybrid TCN - SDN on packet flow and application running on a computer in a computer network. The results showed that the 58,956.3 bytes application running in traditional computer network with 512 packets sent across the network, i.e. A2N0P5, had the most negligible throughput of 14.14 GB while the 65,507 bytes application running in traditional computer network with 32 packets sent across the network, i.e. A1N0P1, had the highest throughput of 18.52 GB. The result indicates that the fewer the packets allowed across the network, the higher the throughput.

Figure 4.1: Results of Effect of Mean of Performance (Bandwidth) of TCN, SDN and Hybrid TCN-SDN on Packet Flow and Application Running on Computer in Computer Network

Figure 4.2: Results of Effect of Mean of Performance (Throughput) of TCN, SDN and Hybrid TCN-SDN on Packet Flow and Application Running on Computer in Computer Network

Figure 4.3 shows the results of the effect of the mean of performance (latency) of Traditional Computer Networking (TCN), Software-Defined Network (SDN), and hybrid TCN - SDN on packet flow and application running on a computer in a computer network. The results showed that the 58,956.3 bytes application running in traditional computer network with 256 packets sent across the network, i.e. A2N0P4, had the least latency of 0.12 ms, while the 10,000 bytes application running in combined traditional computer network and software-defined network with 64 packets sent across the network, i.e. A5N2P2, had highest latency of 0.55 ms. This result indicates that a

combined traditional computer network and software-defined network may add to an increase in delay in network performance as they are not in the homogeneous networks.

Figure 4.4 shows the effect of the mean of performance (jitter) of Traditional Computer Networking (TCN), Software-Defined Network (SDN), and hybrid TCN - SDN on packet flow and applications running on a computer in a computer network. The results showed that the 10,000 bytes application running in traditional computer network with 32 packets sent across the network, i.e. A5N0P1, had the least jitter of 0.001 ms, while the 52405.6 bytes application running in combined traditional computer network and software-defined network with 64 packets sent across the network, i.e. A3N2P2 had the highest jitter of 0.008 ms. The result indicates that a non-homogeneous computer network may increase the delay in the performance of the computer network.

Figure 4.3: Results of Effect of Mean of Performance (Latency) of TCN, SDN and Hybrid TCN-SDN on Packet Flow and Application Running on Computer in Computer Network

Figure 4.4: Results of Effect of Mean of Performance (Jitter) of TCN, SDN and Hybrid TCN-SDN on Packet Flow and Application Running on Computer in Computer Network

4.2.2 ANOVA and DnMR Test of the performance (of the bandwidth, throughput, latency, and jitter) of Traditional Computer Networking (TCN), Software-Defined Network (SDN), and hybrid TCN-SDN on packet flow and application running on a computer in a computer network

Table 4.1 shows the ANOVA results of the effect of performance (latency, bandwidth, throughput, and jitter) of Traditional Computer Networking (TCN), Software-Defined Network (SDN), and hybrid TCN - SDN on packet flow and application running on a computer in a computer network. The results showed that latency, bandwidth, and throughput had a significant effect on the packets

sent across the network, network type, and application running on a computer in a computer network. In contrast, jitter had a partially significant effect on the packets sent across the network, network type, and application running on a computer in a computer network.

Table 4.2 shows the result of the separation of means using Duncan's new multiple range test (DnMR Test) for the effect of performance (latency, bandwidth, throughput, and jitter) of Traditional Computer Networking (TCN), Software-Defined Network (SDN) and hybrid TCN - SDN on packet flow and application running on a computer in a computer network. The results revealed that the effect of applications, i.e. 65,507 bytes (A1), 58956.3 bytes (A2), 52405.6 bytes (A3), 45854.9 bytes (A4) and 10,000 bytes (A5) running on a computer; the number of packets in the byte, i.e. 32 (P1), 64 (P2), 128 (P3), 256 (P4) and 512 (P5) sent across the network and the type of network, i.e Traditional Computer Networking (N0), Software-Defined Network (N1) and hybrid TCN - SDN (N2) had significant influence on latency, bandwidth, throughput and jitter as the means with there letters are not perfectly the same vertically as shown in Table 4.2.

Table 4.1: ANOVA Results of Effect of Performance (latency, bandwidth, throughput and jitter) of Traditional Computer Networking (TCN), Software-Defined Network (SDN) and Hybrid TCN - SDN on Packet Flow and Application Running on Computer in Computer Network

Response	**S. V.**	**Sum Sq**	**Df**	**F value**	**Pr(>F)**
Latency	A	0.19	4	6.89	0.01*
	P	0.74	4	27.43	0.01*
	N	1.38	2	102.91	0.01*
	A x P	0.49	16	4.51	0.01*
	A x N	0.34	8	6.34	0.01*
	P x N	0.68	8	12.64	0.01*
	A x P x N	0.94	32	4.37	0.01*
	Residuals	4.52	675		

Bandwidth (Gbps)	A	20.61	4	81.71	0.01*
	P	1.8	4	7.11	0.01*
	N	42.29	2	335.26	0.01*
	A x P	6.04	16	5.99	0.01*
	A x N	22.41	8	44.42	0.01*
	P x N	4.84	8	9.59	0.01*
	A x P x N	11	32	5.46	0.01*
	Residuals	42.57	675		
Throughput (GB)	A	58.61	4	45.35	0.01*
	P	11.19	4	8.66	0.01*
	N	121.13	2	187.44	0.01*
	A x P	30.52	16	5.91	0.01*
	A x N	83.16	8	32.17	0.01*
	P x N	32.47	8	12.57	0.01*
	A x P x N	44.09	32	4.27	0.01*
	Residuals	218.11	675		
Jitter	A	0.01	4	1.73	0.15ns
	P	0.01	4	2.35	0.06ns
	N	0.01	2	391.09	0.01*
	A x P	0.01	16	3.01	0.01*
	A x N	0.01	8	2.96	0.01*
	P x N	0.01	8	1.4	0.20ns
	A x P x N	0.01	32	2.24	0.01*
	Residuals	0.01	675		

where * = significant different , ns = not significant different

Table 4.2: Duncan's New Multiple Range Test Results for the Effect of Performance (latency (ms), bandwidth (Gbps), throughput (GB) and jitter (ms)) of Traditional Computer Networking (TCN), Software-Defined Network (SDN) and Hybrid TCN - SDN on Packet Flow and Application Running on Computer in Computer Network

T	Latency	Bandwidth	Throughput	Jitter
A1	0.18 ± 0.11b	9.67 ± 0.48a	16.88 ± 0.85a	0.005 ± 0.003b
A2	0.18 ± 0.10b	9.45 ± 0.33b	16.36 ± 1.14b	0.005 ± 0.003ab
A3	0.20 ± 0.12a	9.29 ± 0.40c	16.21 ± 0.69c	0.005 ± 0.003ab
A4	0.17 ± 0.03b	9.22 ± 0.46d	16.09 ± 0.81c	0.005 ± 0.002a
A5	0.21 ± 0.15a	9.26 ± 0.40cd	16.19 ± 0.69c	0.005 ± 0.002ab
P1	0.24 ± 0.19a	9.47 ± 0.47a	16.55 ± 0.82a	0.005 ± 0.002b
P2	0.20 ± 0.12b	9.38 ± 0.43b	16.37 ± 0.73b	0.005 ± 0.003a
P3	0.17 ± 0.05c	9.37 ± 0.44b	16.35 ± 0.78b	0.005 ± 0.002ab
P4	0.16 ± 0.03c	9.33 ± 0.54b	16.29 ± 0.94bc	0.005 ± 0.002ab
P5	0.16 ± 0.05c	9.35 ± 0.34b	16.18 ± 1.12c	0.005 ± 0.002b
N0	0.15 ± 0.03b	9.54 ± 0.42a	16.57 ± 1.06b	0.002 ± 0.001c
N1	0.16 ± 0.01b	9.56 ± 0.27a	16.69 ± 0.48a	0.006 ± 0.002a
N2	0.25 ± 0.17a	9.04 ± 0.43b	15.78 ± 0.75c	0.006 ± 0.002b

Means with the same letter vertically are not significantly different.

where T=Treatment, application runnig in computer on the network A1 = 65,507 bytes, A2 = 58956.3 bytes, A3 = 52405.6 bytes, A4 = 45854.9 bytes, A5 = 10,000 bytes; number of packet sent across network P1 = 32, P2 = 64, P3 = 128, P4 = 256, P5 = 512; the type of network used N0 = traditional computer network, N1 = Software-Defined Network, N2 = hybrid TCN - SDN

4.3 Effect of Background-Traffic and Bandwidth-Limit on the Performance of Software-Defined Network (SDN)

This section presents results and discussion on the influence of Background-Traffic and Bandwidth-

Limit on the performance (latency, bandwidth, throughput, jitter, and datagram loss) of Software-Defined Network (SDN) while downloading email using Thunderbird and web browser as well as getting a website index page through the Internet using other PCs. The section response to the research question raised that why is the influence of background traffic and bandwidth limits on the performance of Software-Defined Network (SDN) significant? The significance of investigating the influence of background traffic and bandwidth limits on the performance of Software-Defined Network (SDN) lies in understanding how these factors impact key performance metrics such as TCP bandwidth, throughput, jitter, datagram loss, latency, and time per 100 packets. According to Figure 4.5, treatments with 64 bytes of traffic to the gateway node (T1) and different bandwidth limits (256M and 1M) showed contrasting TCP bandwidth effects, indicating the critical role of bandwidth limits in optimizing network performance. Higher bandwidth limits were associated with higher TCP bandwidth, throughput (Figure 4.6), and reduced jitter (Figure 4.7), while also impacting datagram loss (Figure 4.8) and latency (Figure 4.9). These findings show the importance of appropriate bandwidth allocation and traffic management in SDN environments, as they can significantly influence the efficiency and reliability of network services, especially under varying traffic loads. The results from the ANOVA and Duncan's New Multiple Range Test further highlight the significant effects and interactions of background traffic and bandwidth limits on these performance metrics, providing valuable insights for optimizing SDN configurations.

4.3.1 Effect of the Mean of Background-Traffic and Bandwidth-Limit on the Latency of Computer Network using Software-Defined Network

Figure 4.5 shows the results of the effect of the mean of Background-Traffic and Bandwidth-Limit on the performance (bandwidth) of Software-Defined Network (SDN). The results showed that treatment with 64 bytes of traffic to the gateway node (T1) and 256M Bandwidth-Limit (U2) on the network had the minor TCP bandwidth effect of 2.01Mbits/sec while treatment with 64 bytes of traffic to the gateway node (T1) and 1M Bandwidth-Limit (U1) on the network had the highest TCP bandwidth effect of 37.94 Mbits/sec. The result showed that the higher the UDP Bandwidth-Limit,

the higher the TCP bandwidth effect while downloading email using Thunderbird, a web browser, and getting a website index page through the Internet using PCs on the network.

The results from Figure 4.6 showed the effect of the mean of Background-Traffic and Bandwidth-Limit on the performance (throughput) of Software-Defined Network (SDN). The results showed that treatment with 64 bytes of traffic to the gateway node (T1) and 256M Bandwidth-Limit (U2) on the network had the least throughput effect of 24.02 Mbits/sec while treatment with 65507 bytes of traffic to the gateway node (T3) and 1M Bandwidth-Limit (U1) on the network had the highest throughput effect of 456.80 Mbits/sec. The result indicates that the higher the UDP Bandwidth-Limit, the higher the throughput effect.

Figure 4.5: Results of Effect of Mean of Background-Traffic and Bandwidth-Limit on the Performance (Bandwidth) of Software-Defined Network (SDN)

Figure 4.6: Results of Effect of Mean of Background-Traffic and Bandwidth-Limit on the Performance (Throughput) of Software-Defined Network (SDN)

Figure 4.7 shows the effect of the mean of Background-Traffic and Bandwidth-Limit on the performance (Jitter) of Software-Defined Network (SDN). The results showed that treatment with 58956 bytes of traffic to the gateway node (T2) and 512M Bandwidth-Limit (U3) on the network had the most negligible jitter of 0.98 ms, while treatment with 64 bytes of traffic to the gateway node (T1) and 512M Bandwidth-Limit (U3) on the network had the highest jitter of 5.66 ms. The results indicate that high UDP Bandwidth-Limit and packet flooding on gateway nodes can cause jitter when downloading emails using Thunderbird, web browsers, and wget, as well as when accessing website index pages using other computers via the internet.

Figure 4.8 shows the results of the effect of the mean of Background-Traffic and Bandwidth-Limit

on the performance (Datagrams loss) of Software-Defined Network (SDN). The results showed that treatment with 58956 bytes of traffic to the gateway node (T2) and 1M Bandwidth-Limit (U1) on the network had the least datagrams loss of 0.002 % while treatment with 64 bytes of traffic to the gateway node (T1) and 512M Bandwidth-Limit (U3) on the network had the highest datagrams loss of 2.024 %. The results indicate that the higher the UDP Bandwidth-Limit, the higher the datagrams loss while downloading email using Thunderbird and web browser and getting a website index page through the Internet using other PCs.

Figure 4.7: Results of Effect of Mean of Background-Traffic and Bandwidth-Limit on the Performance (Jitter) of Software-Defined Network (SDN)

Figure 4.8: Results of Effect of Mean of Background-Traffic and Bandwidth-Limit on the Performance (Datagrams-Loss) of Software-Defined Network (SDN)

Figure 4.9 shows the results of the effect of the mean of Background-Traffic and Bandwidth-Limit on the performance (latency) of Software-Defined Network (SDN). The results showed that treatment with 58956 bytes of traffic to the gateway node (T2) and 1M Bandwidth-Limit (U1) on the network had the least latency of 20.67 ms, while treatment with 64 bytes of traffic to the gateway node (T1) and 256M Bandwidth-Limit (U2) on the network had the highest latency of

62.43 ms.

Figure 4.10 shows the results of the effect of the mean of Background-Traffic and Bandwidth-Limit on the performance (’time per 100 packets ’) of Software-Defined Network (SDN). The results showed that treatment with 64 bytes of traffic to the gateway node (T1) and 512M Bandwidth-Limit (U3) on the network had the least ’time per 100 packets’ of 99134.60 ms while treatment with 58956 bytes of traffic to the gateway node (T2) and 512M Bandwidth-Limit (U3) on the network had the highest ’ time per 100 packets ’ of 99228.00 ms. The result indicates that the higher the traffic to the gateway node, the higher the ’ time per 100 packets ’ experienced.

Figure 4.9: Results of Effect of Mean of Background-Traffic and Bandwidth-Limit on the Performance (Latency) of Software-Defined Network (SDN)

Figure 4.10: Results of Effect of Mean of Background-Traffic and Bandwidth-Limit on the Performance (Time/100packets) of Software-Defined Network (SDN)

4.3.2 ANOVA on the Effect of Background-Traffic and Bandwidth-Limit on Performance (TCP bandwidth, throughput, jitter, datagrams loss, latency, 'time per 100 packets') of Software-Defined Network

Table 4.3 shows the ANOVA on the effect of Background-Traffic and Bandwidth-Limit on performance (TCP bandwidth, throughput, jitter, datagrams loss, latency, and 'time per 100 packets') of Software-Defined Network. The results showed that both Background-Traffic and Bandwidth-Limit had significant effects on TCP bandwidth, throughput, and latency. The

Bandwidth-Limit only had a significant effect on jitter and datagram loss. Also, Background-Traffic had no significant effect on jitter, datagram loss, and 'time per 100 packets'. Interaction between Background-Traffic and Bandwidth-Limit had a significant effect on bandwidth. In contrast, the interaction between Background-Traffic and Bandwidth-Limit had no significant effect on throughput, jitter, datagram loss, latency, and 'time per 100 packets'.

Table 4.3: ANOVA results of effect of Background-Traffic and Bandwidth-Limit on data rate of

transfer of computer network using Software-Defined Network

Response	**S. V.**	**Sum Sq**	**Df**	**F value**	**Pr(>F)**
Bandwidth	T	15.7	2	7.78	0.01*
	U	11483.4	2	5680.89	0.01*
	T x U	16.3	4	4.04	0.01*
	Residuals	36.4	36		
Throughput	T	3166	2	9.06	0.01*
	U	1675087	2	4792.94	0.01*
	T x U	1335	4	1.91	0.13ns
	Residuals	6291	36		
Jitter	T	0.86	2	0.09	0.92ns
	U	41.43	2	4.29	0.03*
	T x U	50.81	4	2.63	0.07ns
	Residuals	106.28	22		
Datagrams loss	T	0.14	2	0.04	0.96ns
	U	22.87	2	6.98	0.01*
	T x U	2.76	4	0.42	0.80ns
	Residuals	58.96	36		
Latency	T	3557.6	2	8.11	0.01*
	U	2482.9	2	5.66	0.01*
	T x U	496.6	4	0.57	0.69ns
	Residuals	6363.4	29		
Time per 100 packets	T	14158	2	2.61	0.09ns
	U	11062	2	2.04	0.15ns
	T x U	10682	4	0.98	0.43ns
	Residuals	97804	36		

4.3.3 Results of Duncan's New Multiple Range Test of the Effect of Background-Traffic and Bandwidth-Limit on Performance (TCP bandwidth, throughput, jitter, datagrams loss, latency, 'time per 100 packets') of Software-Defined Network

Table 4.4 shows the result of the separation of means using Duncan's new multiple range test for the effect of Background-Traffic and Bandwidth-Limit on performance (TCP bandwidth, throughput, jitter, datagrams loss, latency, 'time per 100 packets') of Software-Defined Network. The results revealed that the effect of Background-Traffic T2 and T3 do not significantly differ, but both differ with T1 on TCP bandwidth and throughput. Background-Traffic T1, T2, and T3 do not significantly differ on jitter and datagrams loss but significantly differ on 'time per 100 packets' (Table 4.4). Bandwidth-Limits U1, U2, and U3 do not significantly differ on 'time per 100 packets' but significantly differ on TCP bandwidth, throughput, jitter, datagrams loss, and latency as the means with their letters are not perfectly the same vertically as shown in Table 4.4.

Table 4.4: Duncan's New Multiple Range Test Results for the Effect of Background-Traffic and Bandwidth-Limit on Throughput of Computer Network using Software-Defined Network

	Bandwidth	Throughput	Jitter	Datagrams loss	Latency	Time per 100.packets
T1	17.35 ± 14.27b	206.99 ± 171.09b	3.06 ± 2.88a	1.55 ± 1.31a	51.90 ± 11.66a	99139.07 ± 27.54b
T2	16.00 ± 15.60a	195.43 ± 190.23a	2.99 ± 2.79a	1.19 ± 1.18a	27.58 ± 16.17b	99182.40 ± 73.12a
T3	16.34 ± 15.44a	197.88 ± 187.14a	2.82 ± 2.18a	1.49 ± 1.25a	43.07 ± 18.87a	99163.47 ± 49.33ab
U1	37.68 ± 0.78a	455.47 ± 9.16a	4.28 ± 2.67a	0.95 ± 0.25b	30.06 ± 8.49b	99141.80 ± 7.84a
U2	3.08 ± 1.38c	37.33 ± 17.77c	1.61 ± 1.26b	1.44 ± 1.61a	42.68 ± 20.29a	99163.00 ± 48.46a
U3	4.55 ± 1.54b	55.67 ± 19.26b	3.01 ± 2.86ab	1.20 ± 1.89a	49.22 ± 21.22a	99180.13 ± 79.69a

Means with the same letter vertically are not significantly different

4.4 Description of Dataflow from Datasets on Traffic from LAN and SDN to Internet due to the Influence of Background-Traffic and Bandwidth-Limit on the Performance of Software-Defined Network (SDN)

This section features the results and discussion on the flow of traffic from LAN and SDN to the

Internet due to the influence of Background-Traffic and Bandwidth-Limit on the performance of Software-Defined Network (SDN). The section responds to the research question about how the data flow features of traffic from LAN and SDN to the Internet are influenced by Background-Traffic and Bandwidth-Limit on Software-Defined Network (SDN) performance. The data flow features from LAN and SDN to the Internet are significantly influenced by background traffic and bandwidth limits, affecting the performance of Software-Defined Network (SDN). The datasets analyzed, ranging from 64 bytes to 65507 bytes of traffic and bandwidth limits from 1M to 512M, reveal various patterns and correlations. The study found that switches (SW), flow time (T), number of packets (PN), and bytes (BN) all play critical roles in determining network performance. The results indicate that configurations such as 64 bytes of traffic and a 1M bandwidth limit (T1U1) can generate high-density data flows, which may impact switch performance. Notably, the density of data flow across different switches indicated the presence of traffic congestion at certain points, with a peak density of up to 16.50% in some switches. Additionally, correlations between parameters such as packet numbers and bytes were identified, highlighting their influence on network efficiency. These findings emphasize the importance of carefully managing data flow and bandwidth in SDN to enhance network performance.

4.4.1 Dataflow Features from the Dataset of 64 bytes of Traffic to the Gateway node and 1M Bandwidth-Limit on the Network (T1U1) on the Performance of Software-Defined Network

Table 4.5 shows the data-flow features of traffic from LAN and SDN to the Internet due to the treatment of 64 bytes (T1) as traffic that floods the gateway node and 1M Bandwidth-Limit (U1) on the network as well as non-stop engaging DNS server (8.8.8.8) using default ping on the performance of SDN. The table revealed that the dataset has a capacity of 10949 rows and 15 columns with 10949 non-null for each of the parameters from the dataset.

Table 4.6 shows the count of the number of elements in 64 bytes of traffic to the gateway node and

1M Bandwidth-Limit on the network (64 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T1U1)) dataset, measure of central tendency (that is mean) and variability (that is standard deviation), minimum, first quartile (25 %), second quartile (50 %), third quartile (75 %) and maximum of dataflow of T1U1 for the switches (SW), duration of packet spent on the flow (T), number of packets (PN) and bytes (BN) on the flow, idle age (IA), input port (PI), flow type from the source (FS) and destination (FD) nodes on performance of SDN.

Figure 4.11 shows the density of the switch (open-flow switch) involved in data flow for 64 bytes of traffic to the gateway node and a 1M Bandwidth-Limit on the network (T1U1). The results showed that the data flow for 64 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T1U1) resulted from 0.00 % density in data flow for all the switches to the highest 16.50 % density in switch 3.

Table 4.5: Dataflow Features for Traffic from LAN and SDN to Internet due to Treatment of 64 bytes (T1) as Traffic that Flood the Gateway Node and 1M Bandwidth-Limit (U1) on the Network

S/N	T1	U1	SW	T	PN	BN	IA	PR	PI	RIFS	ANSS	FS	RIFD	ANSD
0	64	1	2	0.01	1	42	0	arp	29	arp_spa=192.168.8.81	192.168.8.81	1	arp_tpa=192.168.8.1	192.168.8.
1	64	1	2	0.01	1	42	0	arp	12	arp_spa=192.168.8.42	192.168.8.42	1	arp_tpa=192.168.8.228	192.168.8.2
2	64	1	3	0.02	1	42	0	arp	24	arp_spa=192.168.8.81	192.168.8.81	1	arp_tpa=192.168.8.1	192.168.8.
3	64	1	3	0.02	1	42	0	arp	29	arp_spa=192.168.8.42	192.168.8.42	1	arp_tpa=192.168.8.228	192.168.8.2
4	64	1	4	0.02	1	42	0	arp	29	arp_spa=192.168.8.42	192.168.8.42	1	arp_tpa=192.168.8.228	192.168.8.2
1094	64	1	5	29.28	91986	1390828320		udp	29	nw_src=192.168.8.42	192.168.8.42	0	nw_dst=192.168.8.228	192.168.8.2
1094	64	1	7	12.76	92	90699	0	tcp	29	nw_src=64.233.184.109	64.233.184.109	0	nw_dst=192.168.8.228	192.168.8.2
1094	64	1	6	14.08	922	2444260	0	tcp	29	nw_src=192.168.8.41	192.168.8.41	0	nw_dst=192.168.8.228	192.168.8.2
1094	64	1	9	12.76	94	92925	0	tcp	2	nw_src=64.233.184.109	64.233.184.109	0	nw_dst=192.168.8.228	192.168.8.2
1094	64	1	8	29.68	94805	14334516	00	udp	29	nw_src=192.168.8.42	192.168.8.42	0	nw_dst=192.168.8.228	192.168.8.2

where Treatment = TR, switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'Raw IP flow, ARP_spa, Nw_src' = RIFS, 'arp & nw

SOURCE 1 for ARP_spa, 0 for Nw_src' = ANSS, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS, 'Raw IP flow, ARP_spa, Nw_dst' = RIFD, 'arp & network SOURCE 1 for ARP_spa, 0 for Nw_dst' =ANSD, and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Table 4.6: Measure of Central Tendency and Variability of Dataflow of 64 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T1U1) on Performance of SDN

	SW	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
mean	4.64	9.9	2163.7	4052385.14	2.51	26.12	0.23	0.23
std	2.59	8.07	9067.27	18804868.25	3.14	8.19	0.42	0.42
min	1	0	0	0	0	2	0	0
25%	2	3.86	1	74	0	29	0	0
50%	4	7.28	6	686	1	29	0	0
75%	7	14.54	27	3038	5	30	0	0
max	9	30.39	95740	144758880	10	30	1	1

Note: count for SW, T, PN, BN, IA, PI, FS and FD = 10949.00

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD.

Figure 4.11: Data-flow Density in Open-flow Switch for 64 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T1U1)

Figure 4.12 shows the pattern results on the features of SW, T, PN, BN, IA, PI, FS, and FD elements in the data flow. T, PN, BN, and IA had the same features. Table 4.7 shows the measures of correlation between pairs of data in 64 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T1U1) dataset. Results of the analysis showed that T, PN, BN, IA, FS, and FD had no apparent relationship with SW, while PI had a weak negative correlation with SW. PN,

BN, IA, and PI had no apparent relationship with T, while FS and FD had a weak negative correlation with T. BN, IA, PI, FS, and FD had no apparent relationship with PN. IA, PI, FS, and FD had no apparent relationship with BN. PI had no apparent relationship with IA, while FS and FD had a positive correlation with IA. FS and FD had no apparent relationship with PI. FS had a strong positive correlation with FD. A further consideration for only numerical features of the T1U1 data frame revealed that PN and BN are strongly correlated, as shown in Figure 4.13.

Figure 4.12: Data-flow across LANs, SDN and Internet for 64 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T1U1)

Table 4.7: Correlation Matrix of Dataflow of 64 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T1U1) on Performance of SDN

	SW	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
SW	1							
T	-0.01	1						
PN	0.08	0.3	1					
BN	0.07	0.28	0.63	1				
IA	0.01	-0.21	-0.18	-0.16	1			
PI	-0.45	-0.01	-0.04	-0.07	0.03	1		
FS	-0.09	-0.31	-0.13	-0.12	0.4	0.01	1	
FD	-0.09	-0.31	-0.13	-0.12	0.4	0.01	1	1

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, ’FLOW, 1 for ARP_spa, 0 for Nw_src’ = FS, ’Raw IP flow, ARP_spa, Nw_dst’ = RIFD and ’FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Figure 4.13: Strongly Correlated Values from the Numerical Features in T1U1 Dataframe

4.4.2 Dataflow Features from a Dataset of 64 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T1U2) on the Performance of Software-Defined Network

Table 4.8 shows the dataflow features of traffic from LAN and SDN to the Internet due to the treatment of 64 bytes (T1) as traffic that floods the gateway node and 256M Bandwidth-Limit (U2) on the network as well as non-stop engaging DNS server (8.8.8.8) using default ping on the performance of SDN. The table revealed that the dataset has a capacity of 7772 rows Õ 15 columns

with 7772 non-null for each parameter from the dataset. Table 4.9 shows the count of the number of elements in 64 bytes of traffic to the gateway node and 256M Bandwidth-Limit on the network (T1U2) dataset, a measure of central tendency (that is mean) and variability (that is standard deviation), minimum, first quartile (25 %), second quartile (50 %), third quartile (75 %) and maximum of dataflow of T1U2 for the Switches (SW), Flow Time (T), Number of Packets (PN) and Number of Bytes (BN) on the flow, Idle Age (IA), Input Port (PI), Flow Source (FS) and Flow Destination (FD) nodes on performance of SDN.

Figure 4.14 shows the density of the switch (open-flow switch) involved in dataflow for 64 bytes of traffic to the gateway node and 256M Bandwidth-Limit on the network (T1U2). The results showed that the dataflow for 64 bytes of traffic to the gateway node and 256M Bandwidth-Limit on the network (T1U2) resulted from 0.00 % density in dataflow for all the switches to the highest 13.50 % density in switches 2 and 3.

Table 4.8: Dataflow Features for Traffic from LAN and SDN to Internet due to Treatment of 64 bytes (T1) as Traffic that Flood the Gateway Node and 256M Bandwidth-Limit (U2) on the Network

	T1	U2	SW	T	PN	BN	IA	PR	PI	RIFS	ANSS	FS	RIFD	ANSD
0	64	256	1	4.68	1	91	4	udp	29	nw_src=192.168.8.228	192.168.8.228	0	nw_dst=8.8.8.8	8.8.8.8
1	64	256	1	4.55	1	141	4	udp	30	nw_src=8.8.8.8	8.8.8.8	0	nw_dst=192.168.8.228	192.168.8.228
2	64	256	1	7.32	1	42	7	arp	30	arp_spa=192.168.8.1	192.168.8.1	1	arp_tpa=192.168.8.50	192.168.8.50
3	64	256	1	7.32	1	42	7	arp	29	arp_spa=192.168.8.50	192.168.8.50	1	arp_tpa=192.168.8.1	192.168.8.1
4	64	256	1	5.17	1	42	5	arp	29	arp_spa=192.168.8.50	192.168.8.50	1	arp_tpa=192.168.8.1	192.168.8.1
7767	64	256	8	26.23	99526	1504833120		udp	29	nw_src=192.168.8.42	192.168.8.42	0	nw_dst=192.168.8.228	192.168.8.228
7768	64	256	5	11.2	996	2741640	0	tcp	29	nw_src=192.168.8.41	192.168.8.41	0	nw_dst=192.168.8.228	192.168.8.228
7769	64	256	9	26.27	99640	1506556800		udp	2	nw_src=192.168.8.42	192.168.8.42	0	nw_dst=192.168.8.228	192.168.8.228
7770	64	256	3	26.44	99741	1508083920		udp	29	nw_src=192.168.8.42	192.168.8.42	0	nw_dst=192.168.8.228	192.168.8.228
7771	64	256	2	26.47	99844	1509641280		udp	12	nw_src=192.168.8.42	192.168.8.42	0	nw_dst=192.168.8.228	192.168.8.228

where Treatment = TR, switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'Raw IP flow, ARP_spa, Nw_src' = RIFS, 'arp & nw

SOURCE 1 for ARP_spa, 0 for Nw_src' = ANSS, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS, 'Raw IP flow, ARP_spa, Nw_dst' = RIFD, 'arp & network SOURCE 1 for ARP_spa, 0 for Nw_dst' =ANSD, and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Table 4.9: Measure of Central Tendency and Variability of Dataflow of 64 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T1U2) on Performance of SDN

	T1	**U2**	**SW**	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
mean	64	256	4.83	7.9	1643.13	2467971.65	3.18	26.23	0.12	0.12
std	0	0	2.59	6.63	10390.45	15718068.7	3.03	8.36	0.32	0.32
min	64	256	1	0	1	42	0	2	0	0
25%	64	256	3	3.44	1	74	0	29	0	0
50%	64	256	5	6.11	2	171	3	29	0	0
75%	64	256	7	9.52	14	1800	6	30	0	0
max	64	256	9	30.02	101899	154071288	10	30	1	1

Note: count for SW, T, PN, BN, IA, PI, FS and FD = 7772.00

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD.

Figure 4.14: Data-flow Density in Open-flow Switch for 64 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T1U2)

Figure 4.15 shows the pattern results on the features of SW, T, PN, BN, IA, PI, FS, and FD elements in the data flow. T, PN, BN, and IA had the same features. Table 4.10 shows the measures of correlation between pairs of data in 64 bytes of traffic to the gateway node and 256M Bandwidth-Limit on the network (T1U2) dataset. Results of the analysis showed that T, PN, BN, IA, FS, and FD had no apparent relationship with SW, while PI had a weak negative correlation with SW. PN, BN, IA, and PI had no apparent relationship with T, while FS and FD had a weak negative correlation with T. BN, IA, PI, FS, and FD had no apparent relationship with PN. IA, PI, FS, and FD had no apparent relationship with BN. PI had no apparent relationship with IA, while FS and FD had a positive correlation with IA. FS and FD had no apparent relationship with PI. FS

had a strong positive correlation with FD. A further consideration for only numerical features of the T1U2 data frame revealed that PN and BN are strongly correlated, as shown in Figure 4.16.

Figure 4.15: Data-flow across LANs, SDN and Internet for 64 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T1U2)

Table 4.10: Correlation Matrix of Dataflow of 64 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the network (T1U2) on Performance of SDN

	SW	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
SW	1							
T	-0.02	1						
PN	0.04	0.3	1					
BN	0.04	0.3	1	1				
IA	0.02	-0.07	-0.17	-0.16	1			
PI	-0.49	-0.02	-0.05	-0.05	0.01	1		
FS	-0.07	-0.15	-0.06	-0.06	0.19	0	1	
FD	-0.07	-0.15	-0.06	-0.06	0.19	0	1	1

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, ’FLOW, 1 for ARP_spa, 0 for Nw_src’ = FS, ’Raw IP flow, ARP_spa, Nw_dst’ = RIFD and ’FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Figure 4.16: Strongly Correlated Values from the Numerical Features in T1U2 Dataframe

4.4.3 Dataflow Features from a Dataset of 64 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T1U3) on the Performance of Software-Defined Network

Table 4.11 shows the data-flow features of traffic from LAN and SDN to the Internet due to the treatment of 64 bytes (T1) as traffic that floods the gateway node and 512M Bandwidth-Limit (U3) on the network as well as non-stop engaging DNS server (8.8.8.8) using default ping on the performance of SDN. The table revealed that the dataset has a capacity of 6488 rows Õ 15 columns with 6488 non-null for each of the parameters from the dataset.

Table 4.12 shows the count of the number of elements in 64 bytes of traffic to the gateway node and 512M Bandwidth-Limit on the network (T1U3) dataset, a measure of central tendency (that is mean) and variability (that is standard deviation), minimum, first quartile (25 %), second quartile (50 %), third quartile (75 %) and maximum of dataflow of T1U3 for the switches (SW), duration of packet spent on the flow (T), number of packets (PN) and bytes (BN) on the flow, idle age (IA), input port (PI), flow type from the source (FS) and destination (FD) nodes on performance of SDN.

Figure 4.17 shows the density of the switch (open-flow switch) involved in data flow for 64 bytes of traffic to the gateway node and 512M Bandwidth-Limit on the network (T1U3). The results showed that the data-flow for 64 bytes of traffic to the gateway node and 512M Bandwidth-Limit on the network (T1U3) resulted from 0.00 % density in data-flow for all the switches to the highest 14.50 % density in switches 2.

Table 4.11: Dataflow Features for Traffic from LAN and SDN to Internet due to Treatment T1 as Traffic that Flood the Gateway Node and U3 on the Network

	T1U3	SW	T	PN	BN	IA	PR	PI	RIFS	ANSS	FS	RIFD	ANSD
0	64	512	1 9.72	1	66	9	tcp	29	nw_src=192.168.8.228	192.168.8.228	0	nw_dst=64.233.184.109	64.233.184.109
1	64	512	1 9.58	1	66	9	tcp	30	nw_src=64.233.184.109	64.233.184.109	0	nw_dst=192.168.8.228	192.168.8.228
2	64	512	1 2.19	1	94	2	udp	29	nw_src=192.168.8.228	192.168.8.228	0	nw_dst=8.8.8.8	8.8.8.8
3	64	512	1 2.19	1	94	2	udp	29	nw_src=192.168.8.228	192.168.8.228	0	nw_dst=8.8.8.8	8.8.8.8
4	64	512	1 2.19	1	97	2	udp	29	nw_src=192.168.8.228	192.168.8.228	0	nw_dst=8.8.8.8	8.8.8.8

6483	64	5124	13.1	998	28880360	tcp	29	nw_src=192.168.8.41	192.168.8.41	0	nw_dst=192.168.8.228	192.168.8.228
6484	64	5125	12.12	998	28880360	tcp	29	nw_src=192.168.8.41	192.168.8.41	0	nw_dst=192.168.8.228	192.168.8.228
6485	64	5126	12.12	998	28880360	tcp	29	nw_src=192.168.8.41	192.168.8.41	0	nw_dst=192.168.8.228	192.168.8.228
6486	64	5127	12.12	998	28880360	tcp	29	nw_src=192.168.8.41	192.168.8.41	0	nw_dst=192.168.8.228	192.168.8.228
6487	64	5128	12.12	998	28880360	tcp	29	nw_src=192.168.8.41	192.168.8.41	0	nw_dst=192.168.8.228	192.168.8.228

where Treatment = TR, switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'Raw IP flow, ARP_spa, Nw_src' = RIFS, 'arp & nw SOURCE 1 for ARP_spa, 0 for Nw_src' = ANSS, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS, 'Raw IP flow, ARP_spa, Nw_dst' = RIFD, 'arp & network SOURCE 1 for ARP_spa, 0 for Nw_dst' =ANSD, and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Table 4.12: Measure of Central Tendency and Variability of Dataflow of 64 bytes of traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T1U3) on Performance of SDN

	T1	**U3**	**SW**	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
mean	64	512	4.76	9.58	2236.14	3353085.92	2.39	26.11	0.19	0.19
std	0	0	2.59	7.97	11420.87	17285619.56	3.13	8.32	0.4	0.4
min	64	512	1	0	1	42	0	2	0	0
25%	64	512	2	2.88	1	74	0	29	0	0
50%	64	512	5	7.62	6	686	1	29	0	0
75%	64	512	7	13.57	29	4803	4	30	0	0

max	64	512	9	30.29	97901	148026312	10	30	1	1

Note: count for SW, T, PN, BN, IA, PI, FS and FD = 6488.00

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD.

Figure 4.17: Data-flow Density in Open-flow Switch for 64 bytes of traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T1U3)

Figure 4.18 shows the pattern results on the features of SW, T, PN, BN, IA, PI, FS, and FD elements in the data flow. T, PN, BN, and IA had the same features. Table 4.13 shows the measures of correlation between pairs of data in 64 bytes of traffic to the gateway node and 512M Bandwidth-Limit on the network (T1U3) dataset. Results of the analysis showed that T, PN, BN,

IA, FS, and FD had no apparent relationship with SW, while PI had a weak negative correlation with SW. PN, BN, IA, and PI had no apparent relationship with T, while FS and FD had a weak negative correlation with T. BN, IA, PI, FS, and FD had no apparent relationship with PN. IA, PI, FS, and FD had no apparent relationship with BN. PI had no apparent relationship with IA, while FS and FD had a positive correlation with IA. FS and FD had no apparent relationship with PI. FS had a strong positive correlation with FD. A further consideration for only numerical features of the T1U3 data frame revealed that PN and BN are strongly correlated, as shown in Figure 4.19.

Figure 4.18: Data-flow across LANs, SDN and Internet for 64 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T1U3)

Table 4.13: Correlation Matrix of Dataflow of 64 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T1U3) on Performance of SDN

	SW	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
SW	1							
T	-0.02	1						
PN	0.06	0.25	1					
BN	0.06	0.25	1	1				
IA	0.01	-0.1	-0.15	-0.15	1			
PI	-0.46	-0.01	-0.06	-0.06	0.02	1		
FS	-0.05	-0.28	-0.1	-0.1	0.32	0	1	
FD	-0.05	-0.28	-0.1	-0.1	0.32	0	1	1

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS, 'Raw IP flow, ARP_spa, Nw_dst' = RIFD and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Figure 4.19: Strongly Correlated Values from the Numerical Features in T1U3 Dataframe

4.4.4 Dataflow Features from a Dataset of 58956 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T2U1) on the Performance of Software-Defined Network

Table 4.14 shows the data-flow features of traffic from LAN and SDN to the Internet due to the treatment of 58956 bytes (T2) as traffic that floods the gateway node and 1M Bandwidth-Limit (U1) on the network as well as non-stop engaging DNS server (8.8.8.8) using default ping on the performance of SDN. The table revealed that the dataset has a capacity of 6022 rows Õ 15 columns with 6022 non-null for each of the parameters from the dataset.

Table 4.15 shows the count of the number of elements in 58956 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T2U1) dataset, a measure of central tendency (that is mean) and variability (that is standard deviation), minimum, first quartile (25 %), second quartile (50 %), third quartile (75 %) and maximum of dataflow of T2U1 for the switches (SW), duration of packet spent on the flow (T), number of packets (PN) and bytes (BN) on the flow, idle age (IA), input port (PI), flow type from the source (FS) and destination (FD) nodes on performance of SDN.

Figure 4.20 shows the density of the switch (open-flow switch) involved in data flow for 58956 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T2U1). The results showed that the data flow for 58956 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T2U1) resulted from 0.00 % density in data flow for all the switches to the highest 15.50 % density in switch 2.

Table 4.14: Dataflow Features for Traffic from LAN and SDN to Internet due to Treatment of 58956 bytes (T2) as Traffic that Flood the Gateway Node and 1M Bandwidth-Limit (U1) on the Network

	T2	U1	SW	T	PN	BN	IA	PR	PI	RIFS	ANSS	FS	RIFD	ANSD
0	58956	1	2	8.48	1	42	8	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.2
1	58956	1	2	7.08	1	42	7	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.2
2	58956	1	2	6.03	1	42	6	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.2
3	58956	1	2	1.19	1	42	1	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.2
4	58956	1	2	5.95	1	42	5	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.2

...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6017	58956	1	8	22.32	91660801	4	tcp	30	nw_src=192.168.8.228	192.168.8.228	0	nw_dst=108.177.15.108	108.177.15.
6018	58956	1	1	21.6	91913859380		icmp	30	nw_src=192.168.8.1	192.168.8.1	0	nw_dst=192.168.8.50	192.168.8.5
6019	58956	1	1	15.04	95913894330		tcp	30	nw_src=108.177.15.108	108.177.15.108	0	nw_dst=192.168.8.228	192.168.8.2
6020	58956	1	1	22.94	96014477760		icmp	30	nw_src=192.168.8.1	192.168.8.1	0	nw_dst=192.168.8.50	192.168.8.5
6021	58956	1	1	29.89	96713943030		tcp	30	nw_src=108.177.15.108	108.177.15.108	0	nw_dst=192.168.8.228	192.168.8.2

where Treatment = TR, switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'Raw IP flow, ARP_spa, Nw_src' = RIFS, 'arp & nw SOURCE 1 for ARP_spa, 0 for Nw_src' = ANSS, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS, 'Raw IP flow, ARP_spa, Nw_dst' = RIFD, 'arp & network SOURCE 1 for ARP_spa, 0 for Nw_dst' =ANSD, and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Table 4.15: Measure of Central Tendency and Variability of Dataflow of 58956 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T2U1) on Performance of SDN

	T2	**U1**	**SW**	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
mean	58956	1	4.73	8.63	966.78	1977122.91	2.87	26.14	0.12	0.12
std	0	0	2.67	6.85	4733.6	13215016.18	3.05	8.43	0.33	0.33
min	58956	1	1	0	0	0	0	2	0	0
25%	58956	1	2	3.53	2	132	0	29	0	0
50%	58956	1	4	6.97	7	645.5	2	29	0	0

75%	58956	1	7	11.03	45	12838	6	30	0	0
max	58956	1	9	30.4	49694	148869550	10	30	1	1

Note: count for SW, T, PN, BN, IA, PI, FS and FD = 6022.00

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD.

Figure 4.20: Data-flow density in open-flow switch for 58956 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T2U1)

Figure 4.21 shows the pattern results on the features of SW, T, PN, BN, IA, PI, FS, and FD

elements in the data flow. T, PN, BN, and IA had the same features. Table 4.16 shows the measures of correlation between pairs of data in 58956 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T2U1) dataset. Results of the analysis showed that T, PN, BN, IA, FS, and FD had no apparent relationship with SW, while PI had a weak negative correlation with SW. PN, BN, IA, and PI had no apparent relationship with T, while FS and FD had a weak negative correlation with T. BN, IA, PI, FS, and FD had no apparent relationship with PN. IA, PI, FS, and FD had no apparent relationship with BN. PI had no apparent relationship with IA, while FS and FD had a positive correlation with IA. FS and FD had no apparent relationship with PI. FS had a strong positive correlation with FD. A further consideration for only numerical features of the T2U1 data frame revealed that PN and BN are strongly correlated, as shown in Figure 4.22.

Figure 4.21: Data-flow across LANs, SDN and Internet for 58956 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T2U1)

Table 4.16: Correlation matrix of dataflow of 58956 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T2U1) on performance of SDN

	SW	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
SW	1							
T	-0.02	1						
PN	0.06	0.25	1					

BN 0.06 0.25 1 1

IA 0.01 -0.1 -0.15 -0.15 1

PI -0.46 -0.01 -0.06 -0.06 0.02 1

FS -0.05 -0.28 -0.1 -0.1 0.32 0 1

FD -0.05 -0.28 -0.1 -0.1 0.32 0 1 1

---

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, ’FLOW, 1 for ARP_spa, 0 for Nw_src’ = FS, ’Raw IP flow, ARP_spa, Nw_dst’ = RIFD and ’FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Figure 4.22: Strongly correlated values from the numerical features in T2U1 dataframe

4.4.5 Dataflow Features from a Dataset of 58956 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T2U2) on the Performance of Software-Defined Network

Table 4.17 shows the data-flow features of traffic from LAN and SDN to the Internet due to the treatment of 58956 bytes (T2) as traffic that floods the gateway node and 256M Bandwidth-Limit (U2) on the network as well as non-stop engaging DNS server (8.8.8.8) using default ping on the

performance of SDN. The table revealed that the dataset has a capacity of 8846 rows Õ 15 columns with 8846 non-null for each of the parameters from the dataset.

Table 4.18 shows the count of the number of elements in 58956 bytes of traffic to the gateway node and 256M Bandwidth-Limit on the network (T2U2) dataset, measure of central tendency (that is mean) and variability (that is standard deviation), minimum, first quartile (25 %), second quartile (50 %), third quartile (75 %) and maximum of dataflow of T2U2 for the switches (SW), duration of packet spent on the flow (T), number of packets (PN) and bytes (BN) on the flow, idle age (IA), input port (PI), flow type from the source (FS) and destination (FD) nodes on performance of SDN.

Figure 4.23 shows the density of the switch (open-flow switch) involved in data flow for 58956 bytes of traffic to the gateway node and the 256M Bandwidth-Limit on the network (T2U2). The results showed that the data flow for 58956 bytes of traffic to the gateway node and 256M Bandwidth-Limit on the network (T2U2) resulted from 0.00 % density in data flow for all the switches to the highest 15.45 % density in switch 3.

Table 4.17: Dataflow Features for Traffic from LAN and SDN to Internet due to Treatment of 58956 bytes (T2) as Traffic that Flood the Gateway Node and 256M Bandwidth-Limit (U2) on the Network

	T2	U2	SW	T	PN	BN	IA	PR	PI	RIFS	ANSS	FS	RIFD	ANSD
0	589562	562		4.04	1	42	4	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.22
1	589562	562		1.01	1	42	1	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.22
2	589562	562		10.08	1	42	10	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.22
3	589562	562		3.24	1	42	3	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.22

4	589562562	4.51	1	42	4	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.22		
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8841	589562565	23.37	983	71263	0	tcp	30	nw_src=192.168.8.228	192.168.8.228	0	nw_dst=108.177.15.108	108.177.15.1		
8842	589562566	23.38	983	71263	0	tcp	30	nw_src=192.168.8.228	192.168.8.228	0	nw_dst=108.177.15.108	108.177.15.1		
8843	589562567	23.39	983	71263	0	tcp	30	nw_src=192.168.8.228	192.168.8.228	0	nw_dst=108.177.15.108	108.177.15.1		
8844	589562565	8.26	99	7350	0	tcp	30	nw_src=192.168.8.228	192.168.8.228	0	nw_dst=108.177.15.108	108.177.15.1		
8845	589562561	27.92	997	15035558	0	icmp	30	nw_src=192.168.8.1	192.168.8.1	0	nw_dst=192.168.8.50	192.168.8.50		

where Treatment = TR, switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'Raw IP flow, ARP_spa, Nw_src' = RIFS, 'arp & nw SOURCE 1 for ARP_spa, 0 for Nw_src' = ANSS, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS, 'Raw IP flow, ARP_spa, Nw_dst' = RIFD, 'arp & network SOURCE 1 for ARP_spa, 0 for Nw_dst' =ANSD, and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Table 4.18: Measure of Central Tendency and Variability of Dataflow of 58956 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T2U2) on Performance of SDN

	T2	**U2**	**SW**	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
mean	58956	256	4.79	9.62	2846.29	4275419.88	2.38	26.16	0.18	0.18
std	0	0	2.59	7.52	14497.72	21974559.8	3.05	8.35	0.39	0.39
min	58956	256	1	0	0	0	0	2	0	0

25%	58956	256	3	3.54	3	198	0	29	0	0
50%	58956	256	5	8.04	10	1241	1	29	0	0
75%	58956	256	7	13.21	47	34806	5	30	0	0
max	58956	256	9	30.45	135554	204957648	10	30	1	1

Note: count for SW, T, PN, BN, IA, PI, FS and FD = 8846.00

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD.

Figure 4.23: Data-flow Density in Open-flow Switch for 58956 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T2U2)

Figure 4.24 shows the pattern results on the features of SW, T, PN, BN, IA, PI, FS, and FD elements in the data flow. T, PN, BN, and IA had the same features. Table 4.19 shows the measures of correlation between pairs of data in 58956 bytes of traffic to the gateway node and the 256M Bandwidth-Limit on the network (T2U2) dataset. Results of the analysis showed that T, PN, BN, IA, FS, and FD had no apparent relationship with SW, while PI had a weak negative correlation with SW. PN, BN, IA, and PI had no apparent relationship with T, while FS and FD had a weak negative correlation with T. BN, IA, PI, FS, and FD had no apparent relationship with PN. IA, PI, FS, and FD had no apparent relationship with BN. PI had no apparent relationship with IA, while FS and FD had a positive correlation with IA. FS and FD had no apparent relationship with PI. FS had a strong positive correlation with FD. A further consideration for only numerical features of the T2U2 data frame revealed that PN and BN are strongly correlated, as shown in Figure 4.25.

Figure 4.24: Data-flow across LANs, SDN and Internet for 58956 bytes of traffic to the gateway node and 256M Bandwidth-Limit on the network (T2U2)

Table 4.19: Correlation Matrix of Dataflow of 58956 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T2U2) on Performance of SDN

	SW	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
SW	1							
T	0	1						
PN	0.06	0.26	1					
BN	0.06	0.26	1	1				

IA	-0.01	0.03	-0.15	-0.15	1			
PI	-0.48	0	-0.05	-0.06	0.01	1		
FS	-0.06	-0.28	-0.09	-0.09	0.34	0	1	
FD	-0.06	-0.28	-0.09	-0.09	0.34	0	1	1

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, ’FLOW, 1 for ARP_spa, 0 for Nw_src’ = FS, ’Raw IP flow, ARP_spa, Nw_dst’ = RIFD and ’FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Figure 4.25: Strongly Correlated Values from the Numerical Features in T2U2 Dataframe

4.4.6 Dataflow Features from a Dataset of 58956 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T2U3) on the Performance of Software-Defined Network

Table 4.20 shows the dataflow features of traffic from LAN and SDN to the Internet due to the treatment of 58956 bytes (T2) as traffic that floods the gateway node and 512M Bandwidth-Limit (U3) on the network as well as non-stop engaging DNS server (8.8.8.8) using default ping on the performance of SDN. The table revealed that the dataset has a capacity of 8075 rows Õ 15 columns

with 8075 non-null for each parameter from the dataset. Table 4.21 shows the count of the number of elements in 58956 bytes of traffic to the gateway node and 512M Bandwidth-Limit on the network (T2U3) dataset, measure of central tendency (that is mean) and variability (that is standard deviation), minimum, first quartile (25 %), second quartile (50 %), third quartile (75 %) and maximum of dataflow of T2U3 for the switches (SW), duration of packet spent on the flow (T), number of packets (PN) and bytes (BN) on the flow, idle age (IA), input port (PI), flow type from the source (FS) and destination (FD) nodes on performance of SDN. Figure 4.26 shows the density of the switch (open-flow switch) involved in dataflow for 58956 bytes of traffic to the gateway node and the 512M Bandwidth-Limit on the network (T2U3). The results showed that the dataflow for 58956 bytes of traffic to the gateway node and 512M Bandwidth-Limit on the network (T2U3) resulted from 0.00 % density in dataflow for all the switches to the highest 14.00 % density in switch 3.

Table 4.20: Dataflow Features for Traffic from LAN and SDN to Internet due to Treatment of 58956 bytes (T2) as Traffic that Flood the Gateway Node and 512M Bandwidth-Limit (U3) on the Network

	T2	U3	SW	T	PN	BN	IA	PR	PI	RIFS	ANSS	FS	RIFD	ANSD
0	58956	5122		4.72	1	42	4	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.22
1	58956	5122		8.57	1	42	8	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.22
2	58956	5122		5.98	1	42	5	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.22
3	58956	5122		7.17	1	42	7	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.22
4	58956	5122		4.14	1	42	4	arp	11	arp_spa=192.1	192.168.8.41	1	arp_tpa=192.16	192.168.8.22

										68.8.41			8.8.228		
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8070	589565125		4.58	97	7462	0	tcp	30		nw_src=192.168.8.228	192.168.8.228	0	nw_dst=108.177.15.108	108.177.15.1		
8071	589565126		4.59	97	7462	0	tcp	30		nw_src=192.168.8.228	192.168.8.228	0	nw_dst=108.177.15.108	108.177.15.1		
8072	589565127		4.61	97	7462	0	tcp	30		nw_src=192.168.8.228	192.168.8.228	0	nw_dst=108.177.15.108	108.177.15.1		
8073	589565121		28.95	974	14161961		tcp	30		nw_src=108.177.15.108	108.177.15.108	0	nw_dst=192.168.8.228	192.168.8.22		
8074	589565123		4.89	993	68286	0	tcp	30		nw_src=192.168.8.228	192.168.8.228	0	nw_dst=192.168.8.41	192.168.8.41		

where Treatment = TR, switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'Raw IP flow, ARP_spa, Nw_src' = RIFS, 'arp & nw SOURCE 1 for ARP_spa, 0 for Nw_src' = ANSS, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS, 'Raw IP flow, ARP_spa, Nw_dst' = RIFD, 'arp & network SOURCE 1 for ARP_spa, 0 for Nw_dst' =ANSD, and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Table 4.21: Measure of Central Tendency and Variability of Dataflow of 58956 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T2U3) on Performance of SDN

	T2	**U3**	**SW**	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
mean	58956	512	4.82	9.88	2544.85	3888226.67	2.68	26.18	0.15	0.15
std	0	0	2.59	7.36	13383.43	20306180.82	3.12	8.37	0.36	0.36
min	58956	512	1	0	0	0	0	2	0	0
25%	58956	512	3	4.32	4	368	0	29	0	0
50%	58956	512	5	8.1	11	1470	1	29	0	0

75%	58956	512	7	13.85	46	22724	5	30	0	0
max	58956	512	9	30.34	127051	192101112	10	30	1	1

Note: count for SW, T, PN, BN, IA, PI, FS and FD = 8075.00

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD.

Figure 4.26: Data-flow density in open-flow switch for 58956 bytes of traffic to the gateway node and 512M Bandwidth-Limit on the network (T2U3)

Figure 4.27 shows the pattern results on the features of SW, T, PN, BN, IA, PI, FS, and FD elements in the data flow. T, PN, BN, and IA had the same features. Table 4.22 shows the measures of correlation between pairs of data in 58956 bytes of traffic to the gateway node and the 512M Bandwidth-Limit on the network (T2U3) dataset. Results of the analysis showed that T, PN, BN, IA, FS, and FD had no apparent relationship with SW, while PI had a weak negative correlation with SW. PN, BN, IA, and PI had no apparent relationship with T, while FS and FD had a weak negative correlation with T. BN, IA, PI, FS, and FD had no apparent relationship with PN. IA, PI, FS, and FD had no apparent relationship with BN. PI had no apparent relationship with IA, while FS and FD had a positive correlation with IA. FS and FD had no apparent relationship with PI. FS had a strong positive correlation with FD. A further consideration for only numerical features of the T2U3 data frame revealed that PN and BN are strongly correlated, as shown in Figure 4.28.

Figure 4.27: Data-flow across LANs, SDN and Internet for 58956 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T2U3)

Table 4.22: Correlation Matrix of Dataflow of 58956 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T2U3) on Performance of SDN

	SW	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
SW	1							
T	0	1						

PN 0.05 0.26 1

BN 0.05 0.26 1 1

IA -0.01 0.06 -0.16 -0.16 1

PI -0.48 -0.01 -0.06 -0.06 0.02 1

FS -0.07 -0.27 -0.08 -0.08 0.26 0 1

FD -0.07 -0.27 -0.08 -0.08 0.26 0 1 1

---

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, ’FLOW, 1 for ARP_spa, 0 for Nw_src’ = FS, ’Raw IP flow, ARP_spa, Nw_dst’ = RIFD and ’FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Figure 4.28: Strongly Correlated Values from the Numerical Features in T2U3 Dataframe

4.4.7 Dataflow Features from a Dataset of 65507 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T3U1) on the Performance of Software-Defined Network

Table 4.23 shows the dataflow features of traffic from LAN and SDN to the Internet due to treatment of 65507 bytes (T3) as traffic that floods the gateway node and 1M Bandwidth-Limit (U1) on the network as well as non-stop engaging DNS server (8.8.8.8) using default ping on the performance of SDN. The table revealed that the dataset has a capacity of 9831 rows Õ 15 columns

with 9831 non-null for each of the parameters from the dataset. Table 4.24 shows the count of the number of elements in 65507 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T3U1) dataset, a measure of central tendency (that is mean) and variability (that is standard deviation), minimum, first quartile (25 %), second quartile (50 %), third quartile (75 %) and maximum of dataflow of T3U1 for the switches (SW), duration of packet spent on the flow (T), number of packets (PN) and bytes (BN) on the flow, idle age (IA), input port (PI), flow type from the source (FS) and destination (FD) nodes on performance of SDN. Figure 4.29 shows the density of the switch (open-flow switch) involved in dataflow for 65507 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T3U1). The results showed that the dataflow for 65507 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T3U1) resulted from 0.00 % density in dataflow for all the switches to the highest 14.50 % density in switch 3.

Table 4.23: Dataflow Features for Traffic from LAN and SDN to Internet due to Treatment of 65507 bytes (T3) as Traffic that Flood the Gateway Node and 1M Bandwidth-Limit (U1) on the Network

	T3	U1	SW	T	PN	BN	IA	PR	PI	RIFS	ANSS	FS	RIFD	ANSI
0	65507	1	2	0.2	1	42	0	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.1 8.8.22 8
1	65507	1	2	0	1	1514	0	tcp	11	nw_src=192.168.8.41	192.168.8.41	0	nw_dst=192.168.8.228	192.1 8.8.22 8
2	65507	1	2	9.27	1	42	9	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.1 8.8.22 8
3	65507	1	2	0.75	1	42	0	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168	192.1

												.8.228	8.8.22 8
4	655071	2	9.81	1	42	9	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168 .8.228	192.1 8.8.22 8
...	...	...	...	...	...	...	...	...	...	...	...	...	...
9826	655071	6	19.9497	7909	6	tcp	30	nw_src=192.168.8.228	192.168.8.228	0		nw_dst=108.177 .15.108	108.1 7.15. 08
9827	655071	7	19.9597	7909	6	tcp	30	nw_src=192.168.8.228	192.168.8.228	0		nw_dst=108.177 .15.108	108.1 7.15. 08
9828	655071	8	19.9697	7909	6	tcp	30	nw_src=192.168.8.228	192.168.8.228	0		nw_dst=108.177 .15.108	108.1 7.15. 08
9829	655071	1	20.99982	14628780		icmp	30	nw_src=192.168.8.1	192.168.8.1	0		nw_dst=192.168 .8.50	192.1 8.8.50
9830	655071	1	20.78983	14654770		icmp	30	nw_src=192.168.8.1	192.168.8.1	0		nw_dst=192.168 .8.50	192.1 8.8.50

where Treatment = TR, switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'Raw IP flow, ARP_spa, Nw_src' = RIFS, 'arp & nw SOURCE 1 for ARP_spa, 0 for Nw_src' = ANSS, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS, 'Raw IP flow, ARP_spa, Nw_dst' = RIFD, 'arp & network SOURCE 1 for ARP_spa, 0 for Nw_dst' =ANSD, and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Table 4.24: Measure of Central Tendency and Variability of Dataflow of 65507 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T3U1) on Performance of SDN

	T3	U1	SW	T	PN	BN	IA	PI	FS	FD
MEAN	65507	1	4.83	10.31	1402.96	3062072.19	2.43	26.17	0.14	0.14
STD	0	0	2.59	7.45	5796.03	16483126.87	3	8.39	0.35	0.35
MIN	65507	1	1	0	0	0	0	2	0	0
25%	65507	1	3	4.64	2	148	0	29	0	0
50%	65507	1	5	8.49	8	872	1	29	0	0
75%	65507	1	7	15.25	30	5882.5	5	30	0	0
MAX	65507	1	9	30.39	48158	142759370	10	30	1	1

Note: count for SW, T, PN, BN, IA, PI, FS and FD = 9831.00

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD.

Figure 4.29: Data-flow Density in Open-flow Switch for 65507 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T3U1)

Figure 4.30 shows the pattern results on the features of SW, T, PN, BN, IA, PI, FS, and FD elements in the data flow. T, PN, BN, and IA had the same features. Table 4.25 shows the measures of correlation between pairs of data in 65507 bytes of traffic to the gateway node and 1M Bandwidth-Limit on the network (T3U1) dataset. Results of the analysis showed that T, PN, BN, IA, FS, and FD had no apparent relationship with SW, while PI had a weak negative correlation with SW. PN, BN, IA, and PI had no apparent relationship with T, while FS and FD had a weak negative correlation with T. BN, IA, PI, FS, and FD had no apparent relationship with PN. IA, PI, FS, and FD had no apparent relationship with BN. PI had no apparent relationship with IA, while FS and FD had a positive correlation with IA. FS and FD had no apparent relationship with PI. FS

had a strong positive correlation with FD. A further consideration for only numerical features of the T3U1 data frame revealed that PN and BN are strongly correlated, as shown in Figure 4.31.

Figure 4.30: Data-flow across LANs, SDN and Internet for 65507 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T3U1)

Table 4.25: Correlation Matrix of Dataflow of 65507 bytes of Traffic to the Gateway Node and 1M

Bandwidth-Limit on the Network (T3U1) on Performance of SDN

	SW	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
SW	1							
T	0	1						
PN	0.06	0.33	1					
BN	0.05	0.25	0.34	1				
IA	-0.01	0	-0.19	-0.15	1			
PI	-0.48	-0.01	-0.02	-0.06	0.01	1		
FS	-0.07	-0.26	-0.1	-0.07	0.34	0	1	
FD	-0.07	-0.26	-0.1	-0.07	0.34	0	1	1

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, ’FLOW, 1 for ARP_spa, 0 for Nw_src’ = FS, ’Raw IP flow, ARP_spa, Nw_dst’ = RIFD and ’FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Figure 4.31: Strongly Correlated Values from the Numerical Features in T3U1 Dataframe

4.4.8 Dataflow Features from a Dataset of 65507 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T3U2) on the Performance of Software-Defined Network

Table 4.26 shows the dataflow features of traffic from LAN and SDN to the Internet due to treatment of 65507 bytes (T3) as traffic that floods the gateway node and 256M Bandwidth-Limit (U2) on the network as well as non-stop engaging DNS server (8.8.8.8) using default ping on the performance of SDN. The table revealed that the dataset has a capacity of 6313 rows Õ 15 columns

with 6313 non-null for each parameter from the dataset. Table 4.27 shows the count of the number of elements in 65507 bytes of traffic to the gateway node and 256M Bandwidth-Limit on the network (T3U2) dataset, a measure of central tendency (that is mean) and variability (that is standard deviation), minimum, first quartile (25 %), second quartile (50 %), third quartile (75 %) and maximum of dataflow of T3U2 for the switches (SW), duration of packet spent on the flow (T), number of packets (PN) and bytes (BN) on the flow, idle age (IA), input port (PI), flow type from the source (FS) and destination (FD) nodes on performance of SDN. Figure 4.32 shows the density of the switch (open-flow switch) involved in dataflow for 65507 bytes of traffic to the gateway node and the 256M Bandwidth-Limit on the network (T3U2). The results showed that the dataflow for 65507 bytes of traffic to the gateway node and 256M Bandwidth-Limit on the network (T3U2) resulted from 0.00 % density in dataflow for all the switches to the highest 15.00 % density in switch 3.

Table 4.26: Dataflow Features for Traffic from LAN and SDN to Internet due to Treatment of 65507 bytes (T3) as Traffic that Flood the Gateway Node and 256M Bandwidth-Limit (U2) on the Network

	T3	U2	SW	T	PN	BN	IA	PR	PI	RIFS	ANSS	FS	RIFD	ANSD	FD
0	65507	256	2	8.56	1	42	8	arp	1	arp_pa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.228	1
1	65507	256	2	4.94	1	42	4	arp	1	arp_pa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.228	1
2	65507	256	2	2.1	1	42	2	arp	1	arp_pa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.228	1
3	65507	256	2	1.09	1	42	1	arp	1	arp_pa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.228	1
4	65507	256	2	6.08	1	42	6	arp	1	arp_pa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.228	1

										68.8.41				68.8.228				
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6308	655072563	8	999660820	tcp	30					nw_rc=192.168.8.228	192.168.8.228	0		nw_dt=192.168.8.41	192.168.8.41	0			
6309	655072564	8.01	999660820	tcp	30					nw_rc=192.168.8.228	192.168.8.228	0		nw_dt=192.168.8.41	192.168.8.41	0			
6310	655072565	8.03	999660820	tcp	30					nw_rc=192.168.8.228	192.168.8.228	0		nw_dt=192.168.8.41	192.168.8.41	0			
6311	655072567	8.06	999660820	tcp	30					nw_rc=192.168.8.228	192.168.8.228	0		nw_dt=192.168.8.41	192.168.8.41	0			
6312	655072568	8.08	999660820	tcp	30					nw_rc=192.168.8.228	192.168.8.228	0		nw_dt=192.168.8.41	192.168.8.41	0			

where Treatment = TR, switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'Raw IP flow, ARP_spa, Nw_src' = RIFS, 'arp & nw SOURCE 1 for ARP_spa, 0 for Nw_src' = ANSS, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS, 'Raw IP flow, ARP_spa, Nw_dst' = RIFD, 'arp & network SOURCE 1 for ARP_spa, 0 for Nw_dst' =ANSD, and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Table 4.27: Measure of Central Tendency and Variability of Dataflow of 65507 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T3U2) on Performance of SDN

	T3	U2	SW	T	PN	BN	IA	PI	FS	FD
MEAN	65507	256	4.73	10.33	3879.8	5888352.26	2.47	26.05	0.21	0.21
STD	0	0	2.58	8.12	16734.25	25357288.06	3.02	8.33	0.41	0.41
MIN	65507	256	1	0	0	0	0	2	0	0
25%	65507	256	2	4.05	1	66	0	29	0	0

50%	65507	256	4	8.16	6	937	1	29	0	0
75%	65507	256	7	15.87	95	16639	5	30	0	0
MAX	65507	256	9	30.21	128862	194839344	10	30	1	1

Note: count for SW, T, PN, BN, IA, PI, FS and FD = 6313.00

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD.

Figure 4.32: Data-flow Density in Open-flow Switch for 65507 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T3U2)

Figure 4.33 shows the pattern results on the features of SW, T, PN, BN, IA, PI, FS, and FD elements in the data flow. T, PN, BN, and IA had the same features. Table 4.28 shows the measures

of correlation between pairs of data in 65507 bytes of traffic to the gateway node and the 256M Bandwidth-Limit on the network (T3U2) dataset. Results of the analysis showed that T, PN, BN, IA, FS, and FD had no apparent relationship with SW, while PI had a weak negative correlation with SW. PN, BN, IA, and PI had no apparent relationship with T, while FS and FD had a weak negative correlation with T. BN, IA, PI, FS, and FD had no apparent relationship with PN. IA, PI, FS, and FD had no apparent relationship with BN. PI had no apparent relationship with IA, while FS and FD had a positive correlation with IA. FS and FD had no apparent relationship with PI. FS had a strong positive correlation with FD. A further consideration for only numerical features of the T3U2 data frame revealed that PN and BN are strongly correlated, as shown in Figure 4.34.

Figure 4.33: Data-flow across LANs, SDN and Internet for 65507 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T3U2)

Table 4.28: Correlation Matrix of Dataflow of 65507 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T3U2) on Performance of SDN

	SW	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
SW	1							
T	0	1						
PN	0.07	0.27	1					

BN 0.07 0.27 1 1

IA 0.01 -0.18 -0.18 -0.18 1

PI -0.45 -0.01 -0.07 -0.07 0.02 1

FS -0.07 -0.31 -0.12 -0.12 0.4 0 1

FD -0.07 -0.31 -0.12 -0.12 0.4 0 1 1

---

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS, 'Raw IP flow, ARP_spa, Nw_dst' = RIFD and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Figure 4.34: Strongly correlated Values from the Numerical Features in T3U2 Dataframe

4.4.9 Dataflow Features from a Dataset of 65507 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T3U3) on the Performance of Software-Defined Network

Table 4.29 shows the dataflow features of traffic from LAN and SDN to the Internet due to the treatment of 65507 bytes (T3) as traffic that floods the gateway node and 512M Bandwidth-Limit (U3) on the network as well as non-stop engaging DNS server (8.8.8.8) using default ping on the performance of SDN. The table revealed that the dataset has a capacity of 5489 rows Õ 15 columns

with 5489 non-null for each parameter from the dataset. Table 4.30 shows the count of the number of elements in 65507 bytes of traffic to the gateway node and 512M Bandwidth-Limit on the network (T3U3) dataset, a measure of central tendency (that is mean) and variability (that is standard deviation), minimum, first quartile (25 %), second quartile (50 %), third quartile (75 %) and maximum of dataflow of T3U3 for the switches (SW), duration of packet spent on the flow (T), number of packets (PN) and bytes (BN) on the flow, idle age (IA), input port (PI), flow type from the source (FS) and destination (FD) nodes on performance of SDN. Figure 4.35 shows the density of the switch (open-flow switch) involved in dataflow for 65507 bytes of traffic to the gateway node and the 512M Bandwidth-Limit on the network (T3U3). The results showed that the dataflow for 65507 bytes of traffic to the gateway node and 512M Bandwidth-Limit on the network (T3U3) resulted from 0.00 % density in dataflow for all the switches to the highest 15.50 % density in switch 3.

Table 4.29: Dataflow Features for Traffic from LAN and SDN to Internet due to Treatment of 65507 bytes (T3) as Traffic that flood the Gateway Node and 512M Bandwidth-Limit (U3) on the Network

	T3	U3	SW	T	PN	BN	IA	PR	PI	RIFS	ANSS	FS	RIFD	ANSD	FD
0	65507	512	2	9.79	1	42	9	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.228	1
1	65507	512	2	1.33	1	42	1	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.228	1
2	65507	512	2	8.95	1	42	8	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.228	1
3	65507	512	2	4.69	1	42	4	arp	11	arp_spa=192.168.8.41	192.168.8.41	1	arp_tpa=192.168.8.228	192.168.8.228	1
4	65507	512	2	1.81	1	42	1	arp	11	arp_spa=192.	192.168.8.41	1	arp_tpa=192.1	192.168.8.228	1

										168.8.41			68.8.228				
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...			
5484	65507	5124	3.42	994706760	tcp	30				nw_src=192.168.8.228	192.168.8.228	0	nw_dst=192.168.8.41	192.168.8.41	0			
5485	65507	5125	3.44	997708740	tcp	30				nw_src=192.168.8.228	192.168.8.228	0	nw_dst=192.168.8.41	192.168.8.41	0			
5486	65507	5127	3.47	997708740	tcp	30				nw_src=192.168.8.228	192.168.8.228	0	nw_dst=192.168.8.41	192.168.8.41	0			
5487	65507	5126	3.46	999710060	tcp	30				nw_src=192.168.8.228	192.168.8.228	0	nw_dst=192.168.8.41	192.168.8.41	0			
5488	65507	5128	3.48	999710060	tcp	30				nw_src=192.168.8.228	192.168.8.228	0	nw_dst=192.168.8.41	192.168.8.41	0			

where Treatment = TR, switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'Raw IP flow, ARP_spa, Nw_src' = RIFS, 'arp & nw SOURCE 1 for ARP_spa, 0 for Nw_src' = ANSS, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS, 'Raw IP flow, ARP_spa, Nw_dst' = RIFD, 'arp & network SOURCE 1 for ARP_spa, 0 for Nw_dst' =ANSD, and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Table 4.30: Measure of Central Tendency and Variability of Dataflow of 65507 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T3U3) on Performance of SDN

	T3	U3	SW	T	PN	BN	IA	PI	FS	FD
MEAN	65507	512	4.73	11.63	4795.68	7357248.39	2.12	25.95	0.25	0.25
STD	0	0	2.58	8.57	18362.55	27851521.39	3.04	8.37	0.43	0.43
MIN	65507	512	1	0	0	0	0	2	0	0
25%	65507	512	2	4.97	1	66	0	29	0	0

50%	65507	512	4	8.84	15	1568	0	29	0	0
75%	65507	512	7	17.8	351	194105	4	30	0	0
MAX	65507	512	9	30.33	128048	193608576	10	30	1	1

Note: count for SW, T, PN, BN, IA, PI, FS and FD = 5489.00

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD.

Figure 4.35: Data-flow Density in Open-flow Switch for 65507 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T3U3)

Figure 4.36 shows the pattern results on the features of SW, T, PN, BN, IA, PI, FS, and FD elements in the data flow. T, PN, BN, and IA had the same features. Table 4.31 shows the measures

of correlation between pairs of data in 65507 bytes of traffic to the gateway node and the 512M Bandwidth-Limit on the network (T3U3) dataset. Results of the analysis showed that T, PN, BN, IA, FS, and FD had no apparent relationship with SW, while PI had a weak negative correlation with SW. PN, BN, IA, and PI had no apparent relationship with T, while FS and FD had a weak negative correlation with T. BN, IA, PI, FS, and FD had no apparent relationship with PN. IA, PI, FS, and FD had no apparent relationship with BN. PI had no apparent relationship with IA, while FS and FD had a positive correlation with IA. FS and FD had no apparent relationship with PI. FS had a strong positive correlation with FD. A further consideration for only numerical features of the T3U3 data frame revealed that PN and BN are strongly correlated, as shown in Figure 4.37.

Figure 4.36: Data-flow across LANs, SDN and Internet for 65507 bytes of Traffic to the gateway Node and 512M Bandwidth-Limit on the Network (T3U3)

Table 4.31: Correlation Matrix of Dataflow of 65507 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T3U3) on Performance of SDN

	SW	**T**	**PN**	**BN**	**IA**	**PI**	**FS**	**FD**
SW	1							
T	0	1						
PN	0.08	0.26	1					

BN 0.08 0.27 1 1

IA -0.01 -0.22 -0.18 -0.18 1

PI -0.44 -0.01 -0.07 -0.08 0.01 1

FS -0.04 -0.42 -0.15 -0.15 0.51 0 1

FD -0.04 -0.42 -0.15 -0.15 0.51 0 1 1

---

where switch = SW, Time(s) = T, packetsNumber = PN, bytesNumber = BN, idleAge = IA, Protocol = PR, inPort = PI, 'FLOW, 1 for ARP_spa, 0 for Nw_src' = FS, 'Raw IP flow, ARP_spa, Nw_dst' = RIFD and 'FLOW, 1 for ARP_spa, 0 for Nw_dst, = FD

Figure 4.37: Strongly correlated values from the numerical features in T3U3 dataframe

4.5 Evaluation of Combined Machine Learning Model using a Dataset on the Influence of Background-Traffic and Bandwidth-Limit on the Performance of Software-Defined Network

This section presents the results and discussion of evaluating combined machine learning model using a dataset that explores the combined influence of Background-Traffic and Bandwidth-Limit on the performance of Software-Defined Network (SDN). Nine datasets were analyzed to understand the flow of traffic from LANs and SDN to the internet. This section provides response to the research question raised: How can machine learning models accurately predict network

performance under varying conditions of background traffic and bandwidth limitations for optimized network performance in real time? The integration of various machine learning models including MLP, KNN, SVM_RBF, DT, and RF significantly impacts the prediction of network performance under varying conditions of background traffic and bandwidth limitations, as indicated by the presented results. The 5Stacked model, combining all these models, consistently demonstrated the highest performance across different metrics, such as Accuracy, MCC, and F1 scores. This suggests that the ensemble approach, leveraging the strengths of multiple models, provide more accurate and reliable predictions for optimizing network performance in real-time. However, the MLP model alone often showed the lowest performance, indicating the importance of model selection and combination in achieving optimal results. These findings underscore the potential of machine learning models in enhancing the efficiency and responsiveness of Software-Defined Network systems under diverse operational conditions.

4.5.1 Evaluation of Combined Machine Learning Model for the Dataset from the Influence of 64 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T1U1) on the Performance of Software-Defined Network

Results in Table 4.32 and Figure 4.38 show that model MLP for T1U1 with the 65 % of the dataset for training had the least performance in MCC (74.89 %), F1 score (80.69 %) and Accuracy score (81.28 %) while 5Stacked model, the model obtained from the stack of KNN; SVM_RBF; DT; RF; MLP operation with the 65 % of the dataset for training had the highest performance 100 % for Accuracy score, MCC and F1 score respectively. The results also show that model SVM_RBF with the 75 % of the dataset for training had the highest performance in Accuracy (99.99 %); F1 (99.99 %); MCC (99.98 %) while the least performance MCC (65.89 %); F1 (65.93 %); Accuracy (70.04 %) occurred in model MLP with the 75 % of the dataset for training. The results also revealed that the least performance MCC (70.28 %), F1 (76.35 %), and Accuracy (77.21 %) also occurred in model MLP with the 85 % of the dataset for training.

Table 4.32: ACC, MCC and F1 Results (%) for Combined KNN, SVM_RBF, DT, RF and MLP Models for T1U1dataset

T1U1	T1U1	65_35			75_25			85_15		
Stacked	Model	ACC	MCC	F1	ACC	MCC	F1	ACC	MCC	F1
2	KNN	97.99	97.29	97.99	98.11	97.45	98.11	98.25	97.64	98.25
	SVM_RBF	99.99	99.98	99.99	99.99	99.98	99.99	99.99	99.99	99.99
	STACK	99.18	98.91	99.19	99.28	99.04	99.28	99.96	99.94	99.96
3	KNN	97.99	97.29	97.99	98.11	97.45	98.11	98.25	97.64	98.25
	SVM_RBF	99.99	99.98	99.99	99.99	99.98	99.99	99.99	99.99	99.99
	DT	89.84	86.66	89.73	89.83	86.64	89.73	90.6	87.55	90.45
	STACK	99.92	99.89	99.92	99.79	99.72	99.79	99.75	99.67	99.75
5	KNN	97.99	97.29	97.99	98.11	97.45	98.11	98.25	97.64	98.25
	SVM_RBF	99.99	99.98	99.99	99.99	99.98	99.99	99.99	99.99	99.99
	DT	89.84	86.66	89.73	89.83	86.64	89.73	90.6	87.55	90.45
	RF	99.9	99.87	99.9	99.99	99.98	99.99	99.98	99.97	99.98
	MLP	81.28	74.89	80.69	70.04	65.89	65.93	77.21	70.28	76.35
	STACK	100	100	100	99.98	99.97	99.98	99.99	99.99	99.99

Figure 4.38: Results on training machine learning models on T1U1 dataset

4.5.2 Evaluation of Combined Machine Learning Model for the Dataset from the Influence of 64 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T1U2) on the Performance of Software-Defined Network

The results from Table 4.33 and Figure 4.39 show that model SVM_RBF for T1U2 with the 65 % of the dataset for training had highest performance of 100 % for Accuracy score, MCC and F1 score respectively. The least performance MCC (36.70 %), F1 (40.73 %), and Accuracy (46.62 %) occurred in model MLP with 65 % of the dataset for training. The least performance MCC

(2.03 %)); Accuracy (26.09 %), and F1 (27.68 %) also occurred in model MLP with 75 % of the dataset for training. Likewise, with the 85 % of the dataset for training, the least performance MCC (22.79 %), F1 (48.03 %), and Accuracy (48.67 %) also occurred in model MLP.

Table 4.33: ACC, MCC and F1 Results (%) for Combined KNN, SVM_RBF, DT, RF and MLP Models for T1U2 Dataset

T1U2	**T1U2**		**65_35**			**75_25**			**85_15**	
Stacked	**Model**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**
2	KNN	98.83	98.2	98.83	98.87	98.25	98.87	98.89	0.98.29	98.89
	SVM_RBF	100	100	100	100	100	100	100	100	100
	STACK	99.72	99.57	99.72	99.52	99.26	99.52	99.71	99.56	99.71
3	KNN	98.83	98.2	98.83	98.87	98.25	98.87	98.89	98.29	98.89
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	85.15	77.62	80.36	85.13	77.59	80.34	85	77.38	80.21
	STACK	99.78	99.66	99.78	99.76	99.63	99.76	99.94	99.91	99.94
5	KNN	98.83	98.2	98.83	98.87	98.25	98.87	98.89	98.29	98.89
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	85.15	77.62	80.36	85.13	77.59	80.34	85	77.38	80.21
	RF	100	100	100	99.95	99.92	99.95	99.94	99.91	99.94
	MLP	46.62	36.7	40.73	26.09	20.3	0.27.68	48.67	22.79	48.03
	STACK	100	100	100	99.98	0.99.97	0.99.98	99.98	99.98	99.98

Figure 4.39: Results on training machine learning models on T1U2 dataset

4.5.3 Evaluation of Combined Machine Learning Model for the Dataset from the Influence of 64 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T1U3) on the Performance of Software-Defined Network

The results from Table 4.34 and Figure 4.40 show that model SVM_RBF for T1U3 with the 65 %, 75 %, 85 % of the dataset for training had the highest performance of 100 % for Accuracy score, MCC and F1 score respectively. The least performance F1 (50.53 %), MCC (52.66 %), and Accuracy (56.23 %) occurred in model MLP with 75 % of the dataset for training. The least performance MCC (29.22 %), F1 (49.47 %), and Accuracy (53.45 %) also occurred in model MLP

with 65 % of the dataset for training. Likewise, with the 85 % of the dataset for training, the least performance MCC (67.45 %), Accuracy (74.48 %), and F1 (75.83 %) also occurred in model MLP under 5Stacked operation.

Table 4.34: ACC, MCC and F1 Results (%) for Combined KNN, SVM_RBF, DT, RF and MLP Models for T1U3 Dataset

T1U3	**T1U3**		**65_35**			**75_25**			**85_15**	
Stacked	**Model**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**
2	KNN	98.77	98.21	98.77	98.77	98.21	98.76	98.8	98.26	98.8
	SVM_RBF	100	100	100	100	100	100	100	100	100
	STACK	99.72	99.59	99.72	99.73	99.61	99.73	99.66	99.5	99.66
3	KNN	98.77	98.21	98.77	98.77	98.21	98.76	98.8	98.26	98.8
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	81.76	74.74	79.29	81.26	73.64	79.13	90.57	86.64	89.99
	STACK	99.69	99.55	99.69	99.75	99.64	99.75	99.85	99.79	99.86
5	KNN	98.77	98.21	98.77	98.77	98.21	98.76	98.8	98.26	98.8
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	81.76	74.74	79.29	81.26	73.64	79.13	90.57	86.64	89.99
	RF	99.98	99.97	99.98	99.88	99.82	99.88	99.93	99.89	99.93
	MLP	53.45	29.22	49.47	56.23	52.66	50.53	74.48	67.45	75.83
	STACK	100	100	100	100	100	100	100	100	100

Figure 4.40: Results on training machine learning models on T1U3 dataset

4.5.4 Evaluation of Combined Machine Learning Model for the Dataset from the Influence of 58956 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T2U1) on the Performance of Software-Defined Network

The results from Table 4.35 and Figure 4.41 show that model SVM_RBF for T2U1 with the 65 %, 75 %, 85 % of the dataset for training had the highest performance of 100 % for Accuracy score, MCC and F1 score respectively. The lowest performance MCC (49.16 %), Accuracy (72.15 %), and F1 (73.72 %) occurred in model MLP with 65 % of the dataset for training. The least performance

MCC (61.45 %), F1 (76.53 %), and Accuracy (83.50 %) also occurred in model MLP under 5Stacked operation with 75 % of the dataset for training. Likewise, with the 85 % of the dataset for training, the least performance MCC (61.69 %), Accuracy (83.65 %), and F1 (76.70 %) also occurred in model MLP.

Table 4.35: ACC, MCC and F1 Results (%) for Combined KNN, SVM_RBF, DT, RF and MLP Models for T2U1 Dataset

T	**T2U1**		**65_35**			**75_25**			**85_15**	
St	**Model**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**
2	KNN	98.9	97.59	98.9	99.18	98.2	99.18	99.04	97.9	99.04
	SVM_RBF	100	100	100	100	100	100	100	100	100
	STACK	99.57	99.05	99.57	99.78	99.52	99.78	99.88	99.74	99.88
3	KNN	98.9	97.59	98.9	99.18	98.2	99.18	99.04	97.9	99.04
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	93.51	85.45	91.69	93.45	85.3	91.62	93.4	85.18	91.56
	STACK	99.9	99.78	99.9	99.91	99.81	99.91	99.84	99.66	99.84
5	KNN	98.9	97.59	98.9	99.18	98.2	99.18	99.04	97.9	99.04
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	93.51	85.45	91.69	93.45	85.3	91.62	93.4	85.18	91.56
	RF	100	100	100	99.98	99.95	99.98	99.96	99.91	99.96
	MLP	72.15	49.16	73.72	83.5	61.45	76.53	83.65	61.69	76.7
	STACK	100	100	100	99.96	99.9	99.96	100	100	100

where T= T2U1, St=Stacked

Figure 4.41: Results on training machine learning models on T2U1 dataset

4.5.5 Evaluation of Combined Machine Learning Model for the Dataset from the Influence of 58956 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T2U2) on the Performance of Software-Defined Network

The results from Table 4.36 and Figure 4.42 show that model SVM_RBF for T2U2 with the 65 %, 75 %, 85 % of the dataset for training had the highest performance of 100 % for Accuracy score, MCC and F1 score respectively. The least performance MCC (42.11 %), Accuracy (59.33 %), and F1 (63.03 %) occurred in model MLP with 65 % of the dataset for training. The least performance MCC (33.99 %), Accuracy (55.13 %), and F1 (56.56 %) also occurred in model MLP with 75 % of the dataset for training. Likewise, with the 85 % of the dataset for training, the least performance

MCC (40.38 %), Accuracy (61.22 %), and F1 (61.22 %) also occurred in model MLP under 5Stacked operation.

Table 4.36: ACC, MCC and F1 Results (%) for Combined KNN, SVM_RBF, DT, RF and MLP Models for T2U2 Dataset

T2U2	**T2U2**		**65_35**			**75_25**			**85_15**	
Stacked	**Model**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**
2	KNN	98.83	97.92	98.83	98.96	98.15	98.95	99.02	98.25	99.01
	SVM_RBF	100	100	100	100	100	100	100	100	100
	STACK	99.74	99.54	99.74	99.37	98.87	99.36	99.76	99.58	99.76
3	KNN	98.83	97.92	98.83	98.96	98.15	98.95	99.02	98.25	99.01
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	91.48	85.14	90.38	91.51	85.21	90.41	91.58	85.34	90.48
	STACK	99.9	99.81	99.9	99.88	99.79	99.88	99.88	99.79	99.88
5	KNN	98.83	97.92	98.83	98.96	98.15	98.95	99.02	98.25	99.01
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	91.48	85.14	90.38	91.51	85.21	90.41	91.58	85.34	90.48
	RF	99.93	99.88	99.93	99.95	99.92	99.95	99.95	99.91	99.95
	MLP	59.33	42.11	63.03	55.13	33.99	56.56	61.22	40.38	61.22
	STACK	99.98	99.97	99.98	99.98	99.97	99.98	99.99	99.98	99.99

Figure 4.42: Results on training machine learning models on T2U2 dataset

4.5.6 Evaluation of Combined Machine Learning Model for the Dataset from the Influence of 58956 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T2U3) on the Performance of Software-Defined Network

The results from Table 4.37 and Figure 4.43 show that the 5Stacked model obtained from the stack of KNN; SVM_RBF, DT, RF; MLP operation for T2U3 with the 65 %, 75 %, 85 % of the dataset for training had the highest performance of 100 %, 99.9 %, 99.9 % for Accuracy score, MCC and F1 score respectively. The least performance MCC (63.22 %), F1 (80.36 %), and Accuracy

(80.68 %) occurred in model MLP with 65 % of the dataset for training. The least performance MCC (40.24 %), Accuracy (64.32 %), and F1 (64.89 %) also occurred in model MLP with 75 % of the dataset for training. Likewise, with the 85 % of the dataset for training, the least performance MCC (37.72 %), Accuracy (61.27 %), and F1 (61.57 %) also occurred in model MLP.

Table 4.37: ACC, MCC and F1 Results (%) for Combined KNN, SVM_RBF, DT, RF and MLP Models for T2U3 Dataset

T2U3	**T2U3**	**65_35**			**75_25**			**85_15**		
Stacked	**Model**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**
2	KNN	98.89	97.81	98.89	99.01	98.04	99	99.13	98.27	99.12
	SVM_RBF	99.98	99.96	99.98	100	100	100	100	100	100
	STACK	99.22	98.46	99.21	99.26	98.53	99.25	99.33	98.68	99.32
3	KNN	98.89	97.81	98.89	99.01	98.04	99	99.13	98.27	99.12
	SVM_RBF	99.98	99.96	99.98	99.98	99.97	99.98	99.99	99.97	99.99
	DT	94.07	88.37	93.48	94.07	88.36	93.48	94.13	88.47	93.55
	STACK	99.81	99.62	99.81	99.69	99.38	99.68	99.66	99.34	99.66
5	KNN	98.89	97.81	98.89	99.01	98.04	99	99.13	98.27	99.12
	SVM_RBF	99.98	99.96	99.98	99.98	99.97	99.98	100	100	100
	DT	94.07	88.37	93.48	94.07	88.36	93.48	94.13	88.47	93.55
	RF	99.9	99.81	99.9	99.97	99.93	99.97	99.96	99.91	99.96
	MLP	80.68	63.22	80.36	64.32	40.24	64.89	61.27	37.72	61.57
	STACK	100	100	100	99.97	99.93	99.97	99.99	99.97	99.99

Figure 4.43: Results on training machine learning models on T2U3 dataset

4.5.7 Evaluation of Combined Machine Learning Model for the Dataset from the Influence of 65507 bytes of Traffic to the Gateway Node and 1M Bandwidth-Limit on the Network (T3U1) on the Performance of Software-Defined Network

The results from Table 4.38 and Figure 4.44 show that the 5Stacked model obtained from the stack of KNN; SVM_RBF, DT, RF; MLP operation for T3U1 with the 65 %, 75 %, 85 % of the dataset for training had the highest performance of 100 % for Accuracy score, MCC and F1 score

respectively. The least performance MCC (51.98 %), F1 (72.62 %), and Accuracy (78.61 %) occurred in model MLP with 75 % of the dataset for training. The least performance MCC (46.73 %), F1 (71.18 %), and Accuracy (76.37 %) also occurred in model MLP under 5Stacked operation with 65 % of the dataset for training. Likewise, with the 85 % of the dataset for training, the least performance MCC (35.77 %), Accuracy (61.40 %), and F1 (67.06 %) also occurred in model MLP.

Table 4.38: ACC, MCC and F1 Results (%) for Combined KNN, SVM_RBF, DT, RF and MLP Models for T3U1 Dataset

T3U1	**T3U1**		**65_35**			**75_25**			**85_15**	
Stacked	**Model**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**
2	KNN	98.92	97.83	98.92	98.98	97.96	98.99	99.03	98.05	99.03
	SVM_RBF	100	100	100	100	100	100	100	100	100
	STACK	99.66	99.31	99.66	99.4	98.81	99.41	99.71	99.42	99.71
3	KNN	98.92	97.83	98.92	98.98	97.96	98.99	99.03	98.05	99.03
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	92.28	84.32	91.73	92.23	84.21	91.66	92.27	84.26	91.62
	STACK	99.69	99.37	99.69	99.7	99.4	99.7	99.69	99.37	99.69
5	KNN	98.92	97.83	98.92	98.98	97.96	98.99	99.03	98.05	99.03
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	92.28	84.32	91.73	92.23	84.21	91.66	92.27	84.26	91.62
	RF	99.94	99.87	99.94	99.97	99.95	99.97	99.93	99.86	99.93
	MLP	76.37	46.73	71.18	78.61	51.98	72.62	61.4	35.77	67.06
	STACK	99.98	99.97	99.98	99.97	99.95	99.97	99.98	99.95	99.98

Figure 4.44: Results on training machine learning models on T3U1 dataset

4.5.8 Results of Evaluation of Combined Machine Learning Model for the Dataset from the Influence of 65507 bytes of Traffic to the Gateway Node and 256M Bandwidth-Limit on the Network (T3U2) on the Performance of Software-Defined Network

The results from Table 4.39 and Figure 4.45 show that the 5Stacked model obtained from the stack of KNN; SVM_RBF, DT, RF; MLP operation for T3U2 with the 65 %, 75 %, 85 % of the dataset for training had the highest performance of 100 % for Accuracy score, MCC and F1 score respectively. The least performance MCC (20.00 %), F1 (34.99 %), and Accuracy (36.00 %) occurred in model MLP with 65 % of the dataset for training. The least performance MCC

(60.85 %), Accuracy (65.57 %), and F1 (66.38 %) also occurred in model MLP with 75 % of the dataset for training. Likewise, with the 85 % of the dataset for training, the least performance MCC (55.33 %), Accuracy (68.65 %), and F1 (68.87 %) also occurred in model MLP.

Table 4.39: ACC, MCC and F1 Results (%) for Combined KNN, SVM_RBF, DT, RF and MLP Models for T3U2 Dataset

T3U12	**T3U2**		**65_35**			**75_25**			**85_15**	
Stacked	**Model**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**
2	KNN	98.05	97.23	98.05	98.23	97.48	98.22	98.27	97.53	98.26
	SVM_RBF	100	100	100	100	100	100	100	100	100
	STACK	99.34	99.07	99.34	99.83	99.76	99.83	99.37	99.11	99.37
3	KNN	98.05	97.23	98.05	98.23	97.48	98.22	98.27	97.53	98.26
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	87.47	82.09	87.38	87.45	82.07	87.38	87.64	82.36	87.53
	STACK	99.78	99.69	99.78	99.79	99.7	99.79	99.76	99.66	99.76
5	KNN	98.05	97.23	98.05	98.23	97.48	98.22	98.27	97.53	98.26
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	87.47	82.09	87.38	87.45	82.07	87.38	87.64	82.36	87.53
	RF	99.98	99.97	99.98	99.94	99.91	99.94	99.94	99.92	99.94
	MLP	36	20	34.99	65.57	60.85	66.38	68.65	55.33	68.87
	STACK	100	100	100	99.98	99.97	99.98	100	100	100

Figure 4.45: Results on training machine learning models on T3U2 dataset

4.5.9 Evaluation of Combined Machine Learning Model for the Dataset from the Influence of 65507 bytes of Traffic to the Gateway Node and 512M Bandwidth-Limit on the Network (T3U3) on the Performance of Software-Defined Network

The results from Table 4.40 and Figure 4.46 show that the SVM_RBF model for T3U3 with the 65 %, 75 %, 85 % of the dataset for training had the highest performance of 100 % for Accuracy score, MCC and F1 score respectively. The least performance MCC (21.58 %), F1 (36.42 %), and Accuracy (38.63 %) occurred in model MLP with 65 % of the dataset for training. The least

performance MCC (50.57 %), Accuracy (60.98 %), and F1 (61.97 %) also occurred in model MLP with 75 % of the dataset for training. Likewise, with the 85 % of the dataset for training, the least performance MCC (55.30 %), F1 (59.47 %), and Accuracy (61.07 %) also occurred in model MLP.

Table 4.40: ACC, MCC and F1 Results (%) for Combined KNN, SVM_RBF, DT, RF and MLP Models for T3U3 Dataset

T3U3	**T3U3**		**65_35**			**75_25**			**85_15**	
Stacked	**Model**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**	**ACC**	**MCC**	**F1**
2	KNN	97.62	96.65	97.61	97.96	97.13	97.96	97.92	97.07	97.92
	SVM_RBF	100	100	100	100	100	100	100	100	100
	STACK	99.13	98.79	99.13	99.15	98.81	99.15	99.16	98.83	99.16
3	KNN	97.62	96.65	97.61	97.96	97.13	97.96	97.92	97.07	97.92
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	94.08	91.69	94	93.37	90.66	93.28	93.48	90.82	93.39
	STACK	99.52	99.33	99.52	99.51	99.32	99.52	99.4	99.16	99.4
5	KNN	97.62	96.65	97.61	97.96	97.13	97.96	97.92	97.07	97.92
	SVM_RBF	100	100	100	100	100	100	100	100	100
	DT	94.08	91.69	9400	93.37	90.66	93.28	93.48	90.82	93.39
	RF	99.89	99.84	99.89	99.98	99.97	99.98	100	100	100
	MLP	38.63	21.58	36.42	60.98	50.57	61.97	61.07	55.3	59.47
	STACK	99.92	99.88	99.92	100	100	100	99.98	99.97	99.98

Figure 4.46: Results on training machine learning models on T3U3 dataset

4.5.10 Combined Machine Learning Models on the Dataset from the Influence of Background-Traffic to the Gateway Node and Bandwidth-Limit on the Network on the Performance of Software-Defined Network

Table 4.41 and Figure 4.47 shows the results of combined models KNN+SVM_RBF (2-StkMdl), KNN + SVM_RBF + DT (3-StkMdl), and KNN+SVM_RBF+DT+RF+MLP (5-StkMdl) with 35 % dataset for the test. The results revealed that the two stack model with 35 % dataset of 5489 rows

for testing having 13.6 ± 1.65 s execution time under the treatment T3U3 (i.e 2-SkMdl_35-T3U3) had the least Accuracy score of 96.46 %. In comparison, the highest Accuracy score of 99.51 % occurred in the five stack model with a 35 % dataset of 9831 rows for testing having 55.10 ± 15.90 s execution time under the treatment T3U1 (i.e., 5-SkMdl_35-T3U1). The results in Table 4.40 revealed that the two stack model with a 35 % dataset of 6022 rows for testing having 25.1 ± 0.48 s execution time under the treatment T2U1 (i.e. 2-SkMdl_35-T2U1) had the least MCC (%) score of 94.85 %. In comparison, the highest MCC (%) score of 99.27 % occurred in the five stack model with a 35 % dataset of 5489 rows for testing having 26.60 ± 0.91 s execution time under the treatment T3U3 (i.e., 5-SkMdl_35-T3U3). The results in Table 4.40 also revealed that the two stack models with a 35 % dataset of 5489 rows for testing having 13.60 ± 1.65 s execution time under the treatment T3U3 (i.e., 2-SkMdl_35-T3U3) had the least F1 score (%) of 96.46 %. In comparison, the highest F1 score (%) of 99.50 % occurred in the five stack model with a 35 % dataset of 9831 rows for testing having 55.10 ± 15.90 s execution time under the treatment T3U1 (i.e 5-SkMdl_35-T3U1).

Table 4.41: Results of combined models KNN+SVM_RBF (2-StkMdl), KNN+SVM_RBF+DT (3-StkMdl) and KNN+SVM_RBF+DT+RF+MLP (5-StkMdl) with 35 % Dataset for Testing

Stacked model	**Rows**	**Exec time (s)**	**Acc (%)**	**MCC (%)**	**F1 (%)**
2-StkMdl_35-T1U1	10949	38.70 ± 7.16	96.87	95.78	96.88
2-StkMdl_35-T1U2	7772	23.80 ± 1.76	96.88	95.16	96.87
2-StkMdl_35-T1U3	6488	19.50 ± 2.88	97.18	95.89	97.17
2-StkMdl_35-T2U1	6022	25.10 ± 0.48	97.63	94.85	97.65
2-StkMdl_35-T2U2	8846	42.80 ± 6.85	98.32	97.00	98.31
2-StkMdl_35-T2U3	8075	37.40 ± 7.19	97.49	95.02	97.48
2-StkMdl_35-T3U1	9831	40.20 ± 3.79	97.47	94.92	97.49
2-StkMdl_35-T3U2	6313	16.90 ± 2.55	96.52	95.04	96.51

2-StkMdl_35-T3U3	5489	13.60 ± 1.65	96.46	95.03	96.46
3-StkMdl_35-T1U1	10949	68.00 ± 18.30	98.46	97.61	98.45
3-StkMdl_35-T1U2	7772	54.20 ± 1.61	97.62	96.54	97.62
3-StkMdl_35-T1U3	6488	53.70 ± 2.24	97.62	96.54	97.62
3-StkMdl_35-T2U1	6022	56.80 ± 8.87	98.2	96.05	98.2
3-StkMdl_35-T2U2	8846	67.00 ± 33.30	98.29	96.95	98.27
3-StkMdl_35-T2U3	8075	63.00 ± 28.70	98.48	96.98	98.47
3-StkMdl_35-T3U1	9831	68.00 ± 42.90	97.91	95.78	97.91
3-StkMdl_35-T3U2	6313	48.30 ± 4.98	97.42	96.33	97.42
3-StkMdl_35-T3U3	5489	39.30 ± 0.98	97.5	96.49	97.5
5-StkMdl_35-T1U1	10949	48.40 ± 13.70	98.96	98.59	98.96
5-StkMdl_35-T1U2	7772	36.20 ± 6.08	99.19	98.75	99.19
5-StkMdl_35-T1U3	6488	29.70 ± 1.71	98.99	98.53	98.99
5-StkMdl_35-T2U1	6022	24.60 ± 3.05	99.1	98.04	99.1
5-StkMdl_35-T2U2	8846	39.60 ± 5.24	99.29	98.74	99.29
5-StkMdl_35-T2U3	8075	36.50 ± 5.78	99.12	98.25	99.11
5-StkMdl_35-T3U1	9831	55.10 ± 15.90	99.51	99	99.5
5-StkMdl_35-T3U2	6313	32.30 ± 3.54	98.55	97.94	98.55
5-StkMdl_35-T3U3	5489	26.60 ± 0.91	99.48	99.27	99.48

Figure 4.47: Results of 35 % T1U1 - T3U3 datasets on testing 2, 3, and 5 Stack Machine learning models

The results from Table 4.42 and Figure 4.48 show that the 5-StkMdl stack model, which combines KNN, SVM_RBF, DT, RF, and MLP, had the highest Accuracy score of 99.49%, MCC of 99.28%, and F1 score of 99.49% with an execution time of 24.50 ± 2.35 seconds under treatment T3U3. On the other hand, the 2-StkMdl model, which combines KNN and SVM_RBF, had the lowest Accuracy score of 97.40%, MCC of 94.78%, and F1 score of 97.42% with an execution time of 47.70 ± 10.90 seconds under treatment T3U1. All models were tested using a 25% dataset.

Table 4.42: Results of combined models KNN+SVM_RBF (2-StkMdl), KNN+SVM_RBF+DT (3-StkMdl) and KNN+SVM_RBF+DT+RF+MLP (5-StkMdl) with 25 % Dataset for Testing

Stacked model	Rows	Exec time (s)	Acc (%)	MCC (%)	F1 (%)
2-StkMdl_25-T1U1	10949	59.30 ± 19.70	97.19	96.20	97.19
2-StkMdl_25-T1U2	7772	41.70 ± 7.47	97.01	95.39	97.02
2-StkMdl_25-T1U3	6488	32.40 ± 0.66	96.98	95.60	96.98
2-StkMdl_25-T2U1	6022	24.90 ± 3.68	97.88	95.35	97.88
2-StkMdl_25-T2U2	8846	44.20 ± 7.96	98.28	96.93	98.27
2-StkMdl_25-T2U3	8075	43.00 ± 8.47	97.62	95.29	97.62
2-StkMdl_25-T3U1	9831	47.70 ± 10.90	97.40	94.78	97.42
2-StkMdl_25-T3U2	6313	22.00 ± 3.46	96.45	94.96	96.45
2-StkMdl_25-T3U3	5489	20.70 ± 1.76	96.43	94.98	96.43
3-StkMdl_25-T1U1	10949	49.30 ± 11.70	97.88	97.14	97.88
3-StkMdl_25-T1U2	7772	32.30 ± 5.81	98.04	96.98	98.04
3-StkMdl_25-T1U3	6488	24.80 ± 3.04	97.41	96.23	97.41
3-StkMdl_25-T2U1	6022	18.80 ± 1.08	98.07	95.77	98.07
3-StkMdl_25-T2U2	8846	44.70 ± 7.14	98.78	97.82	98.77
3-StkMdl_25-T2U3	8075	42.10 ± 7.06	98.32	96.66	98.31
3-StkMdl_25-T3U1	9831	45.90 ± 8.64	97.88	95.73	97.88
3-StkMdl_25-T3U2	6313	19.60 ± 3.40	97.40	96.31	97.40
3-StkMdl_25-T3U3	5489	20.30 ± 2.13	97.31	96.21	97.30
5-StkMdl_25-T1U1	10949	62.00 ± 17.80	99.27	99.01	99.27
5-StkMdl_25-T1U2	7772	44.40 ± 5.38	99.18	98.73	99.18
5-StkMdl_25-T1U3	6488	36.20 ± 1.52	98.58	97.94	98.57
5-StkMdl_25-T2U1	6022	37.50 ± 7.95	99.07	97.97	99.08
5-StkMdl_25-T2U2	8846	53.90 ± 8.83	99.37	98.87	99.37

5-StkMdl_25-T2U3	8075	52.10 ± 9.83	99.46	98.92	99.45
5-StkMdl_25-T3U1	9831	53.90 ± 12.70	99.47	98.93	99.47
5-StkMdl_25-T3U2	6313	28.70 ± 3.89	98.8	98.29	98.8
5-StkMdl_25-T3U3	5489	24.50 ± 2.35	99.49	99.28	99.49

Figure 4.48: Results of 25 % T1U1 - T3U3 datasets on testing 2, 3, and 5 Stack Machine learning models

The results of the experiments conducted using combined models, KNN+SVM_RBF (2-StkMdl), KNN+SVM_RBF+DT (3-StkMdl), and KNN+SVM_RBF+DT+RF+MLP (5-StkMdl) with a 15% dataset for testing, are presented in Table 4.43 and Figure 4.49. The results indicate that the stack model 5-SkMdl achieved the highest accuracy score of 99.66%, MCC score of 99.32%, F1 score of

99.66%, and an execution time of 224.00 ± 53.20 s under treatment T3U1. On the other hand, the model 2-SkMdl achieved the least accuracy score of 96.20%, MCC score of 94.59%, F1 score of 96.19%, and an execution time of 224.00 ± 53.20 s under the treatment T3U2.

Table 4.43: Results of combined models KNN+SVM_RBF (2-StkMdl), KNN+SVM_RBF+DT (3-StkMdl) and KNN+SVM_RBF+DT+RF+MLP (5-StkMdl) with 15 % Dataset for Testing

Stacked model	**Rows**	**Exec time (s)**	**Acc (%)**	**MCC (%)**	**F1 (%)**
2-StkMdl_15-T1U1	10949	63.00 ± 14.20	96.90	95.81	96.91
2-StkMdl_15-T1U2	7772	44.10 ± 6.67	97.51	96.15	97.51
2-StkMdl_15-T1U3	6488	29.60 ± 3.69	97.33	96.11	97.33
2-StkMdl_15-T2U1	6022	57.00 ± 5.67	98.23	96.11	98.23
2-StkMdl_15-T2U2	8846	83.00 ± 26.00	98.19	96.77	98.18
2-StkMdl_15-T2U3	8075	82.00 ± 38.20	98.18	96.40	98.18
2-StkMdl_15-T3U1	9831	92.00 ± 34.80	97.76	95.54	97.79
2-StkMdl_15-T3U2	6313	51.80 ± 3.16	96.20	94.59	96.19
2-StkMdl_15-T3U3	5489	43.30 ± 1.16	96.84	95.56	96.84
3-StkMdl_15-T1U1	10949	58.80 ± 16.20	97.69	96.88	97.70
3-StkMdl_15-T1U2	7772	41.00 ± 6.92	98.63	97.88	98.63
3-StkMdl_15-T1U3	6488	29.10 ± 4.03	98.05	97.16	98.04
3-StkMdl_15-T2U1	6022	28.50 ± 4.46	98.78	97.32	98.78
3-StkMdl_15-T2U2	8846	52.10 ± 12.90	98.72	97.72	98.72
3-StkMdl_15-T2U3	8075	46.30 ± 7.06	98.84	97.71	98.84
3-StkMdl_15-T3U1	9831	56.30 ± 7.26	98.17	96.31	98.17
3-StkMdl_15-T3U2	6313	27.50 ± 4.96	97.89	97.00	97.89
3-StkMdl_15-T3U3	5489	20.10 ± 3.21	97.69	96.76	97.69
5-StkMdl_15-T1U1	10949	240.00 ± 60.00	99.15	98.85	99.15

5-StkMdl_15-T1U2	7772	198.00± 42.70	99.23	98.81	99.22
5-StkMdl_15-T1U3	6488	165.00 ± 32.00	98.56	97.91	98.56
5-StkMdl_15-T2U1	6022	140.00 ± 11.60	99.34	98.54	99.34
5-StkMdl_15-T2U2	8846	221.00 ± 46.30	99.17	98.54	99.17
5-StkMdl_15-T2U3	8075	208.00 ± 33.60	99.17	98.37	99.17
5-StkMdl_15-T3U1	9831	224.00 ± 53.20	99.66	99.32	99.66
5-StkMdl_15-T3U2	6313	122.00 ± 11.70	99.26	98.95	99.26
5-StkMdl_15-T3U3	5489	102.00 ± 7.03	99.15	98.81	99.15

Figure 4.49: Results of 15 % T1U1 - T3U3 datasets for testing 2, 3, and 5 Stack Machine learning models

4.6 Reflection

The findings of the study explore the impact of different networking approaches on the performance

of packet flow and applications in a computer network. On the bandwidth; the 10,000-byte application in a hybrid TCN - SDN (A5N2P4) had the lowest bandwidth usage (8.46 Gbps), the 65,507 bytes application in a traditional computer network (A1N0P1) had the highest bandwidth usage (10.62 Gbps), the trend suggests that fewer packets sent across the network result in higher bandwidth usage. On the throughput; the 58,956.3 bytes application in a traditional computer network (A2N0P5) had the lowest throughput (14.14 GB), the 65,507 bytes application in a traditional computer network (A1N0P1) had the highest throughput (18.52 GB), similar to bandwidth, fewer packets across the network correlate with higher throughput. On the latency; the 58,956.3 bytes application in a traditional computer network (A2N0P4) had the lowest latency (0.12 ms), the 10,000 bytes application in a hybrid TCN - SDN (A5N2P2) had the highest latency (0.55 ms). The results suggest that a combination of traditional and SDN networks may lead to increased latency. On the jitter; the 10,000 bytes application in a traditional computer network (A5N0P1) had the lowest jitter (0.001 ms), the 52,405.6 bytes application in a hybrid TCN - SDN (A3N2P2) had the highest jitter (0.008 ms). The findings indicate that a non-homogeneous computer network may increase jitter, impacting network performance. The ANOVA Results show Latency, bandwidth, and throughput significantly affect packets sent, network type, and application. Jitter has a partially significant effect on these factors. Duncan's Multiple Range Test revealed that Applications, the number of packets, and the type of network significantly influence latency, bandwidth, throughput, and jitter. The study at this stage highlights the trade-offs between different networking approaches concerning bandwidth usage, throughput, latency, and jitter. The results emphasize the importance of considering network type, packet size, and application characteristics in optimizing computer network performance.

On the study of the impact of Background-Traffic and Bandwidth-Limits on latency, throughput, jitter, datagrams loss, and other performance metrics of Software-Defined Network (SDN) the results demonstrated that higher UDP Bandwidth-Limits corresponded to higher TCP bandwidth

effects, particularly noticeable during email downloads and web browsing. The subsequent results presented similar trends for throughput, jitter, datagrams loss, and latency. Notably, higher UDP Bandwidth-Limits were associated with increased throughput and datagrams loss, while the relationship with latency varied. The ANOVA results confirmed the significance of Background-Traffic and Bandwidth-Limits on TCP bandwidth, throughput, and latency. Additionally, Duncan's multiple range test detailed specific differences between Background-Traffic types and Bandwidth-Limits for each performance metric, providing a comprehensive understanding of their individual and interactive effects on SDN performance.

The study also explores data flow features in a Software-Defined Network (SDN) environment across diverse traffic and bandwidth conditions. The dataset comprises scenarios representing combinations of traffic types and Bandwidth-Limits, each with 15 parameters. Notably, all datasets exhibit non-null values for specific rows. Dataflow features include switches, duration of packet flow, number of packets, bytes on the flow, idle age, input port, flow type from source, and flow type to destination. Specific analyses for different datasets reveal insights into the relationship between dataflow features under specific conditions. For instance, the T1U2 dataset investigates 64 bytes of traffic to the gateway node with a 256M Bandwidth-Limit, showing a density of switches ranging from 0.00% to 13.50%. Correlation analyses consistently highlight strong relationships, particularly between the number of packets and bytes in the flow. Similar analyses are conducted for other datasets, varying traffic types and Bandwidth-Limits. Overall, the findings emphasize the influence of traffic type and Bandwidth-Limit on dataflow features in an SDN environment. The consistent strong correlation between the number of packets and bytes suggests a potential intrinsic relationship. These results contribute valuable insights into how different traffic and bandwidth conditions impact the performance and characteristics of data flow in a Software-Defined Network environment.

In the evaluation of a stack machine learning model for different datasets reflecting the influence of

traffic and bandwidth on Software-Defined Network (SDN), various models and stacking configurations were tested. For instance, in T1U1, the 5Stacked model, comprised of KNN, SVM_RBF, DT, RF, and MLP, demonstrated optimal performance with 100% accuracy, MCC, and F1 score. However, the MLP model under 5Stacked with 35% testing data had the least performance across metrics. Similar trends were observed in T1U2 and T1U3, with SVM_RBF consistently outperforming, and MLP under 5Stacked consistently exhibiting lower metrics, especially with smaller testing datasets. The evaluation extended to T2U1, T2U2, and T2U3, where SVM_RBF again showed superior performance, while MLP under 5Stacked consistently had the lowest metrics. The pattern persisted in T3U1, T3U2, and T3U3, with SVM_RBF excelling and MLP under 5Stacked consistently performing poorly, particularly with reduced testing datasets. These results underscore the sensitivity of the model's performance to both the traffic conditions and Bandwidth-Limits in SDN environments.

The combined machine learning models, including 2-StkMdl (KNN+SVM_RBF), 3-StkMdl (KNN+SVM_RBF+DT), and 5-StkMdl (KNN+SVM_RBF+DT+RF+MLP), were evaluated on a dataset assessing the impact of Background-Traffic to the gateway node and Bandwidth-Limit on Software-Defined Network performance. Results revealed that the 5-StkMdl with a 35% dataset had the highest Accuracy score of 99.51%, MCC score of 99.27%, and F1 score of 99.50%, with an execution time of 55.10 ± 15.90 seconds under treatment T3U1. Conversely, the 2-StkMdl with a 35% dataset exhibited the lowest Accuracy score of 96.46%, MCC score of 94.85%, and F1 score of 96.46%, with an execution time of 13.60 ± 1.65 seconds under treatment T3U3. Results further highlighted the superiority of the 5-StkMdl, achieving the highest scores across all metrics (Accuracy: 99.49%, MCC: 99.28%, F1: 99.49%) with an execution time of 24.50 ± 2.35 seconds under treatment T3U3. In addition, experiments with a 15% dataset showed the 5-StkMdl with an accuracy of 99.66%, MCC of 99.32%, and F1 score of 99.66%, outperforming the 2-StkMdl with an accuracy of 96.20%, MCC of 94.59%, and F1 score of 96.19% under treatment T3U1, both

having an execution time of 224.00 ± 53.20 seconds.

The 5-stacked model, integrating KNN, SVM_RBF, DT, RF, and MLP, achieved optimal performance with 100% accuracy, MCC, and F1 score in certain scenarios, notably outperforming previous SVM classification algorithms used in SDN environments, which achieved an average accuracy of 95.24%[3]. MLP component underperformed consistently across different scenarios, especially with smaller testing datasets. There was a noticeable impact of dataset size on model performance, with smaller testing datasets often resulting in lower metrics across all models, but particularly affecting MLP in the 5Stacked configuration. The 5-stacked model (KNN, SVM_RBF, DT, RF, MLP) achieved 100% accuracy, MCC, and F1 score, notably outperforming previous methods in detecting DDoS attacks in SDN that classify and identify attack flows with an accuracy rate of 99%[4]. Also The 5-stacked model (KNN, SVM_RBF, DT, RF, MLP) achieved perfect scores of 100% in accuracy, MCC, and F1, outperforming the model from the fourteen NASA datasets with accuracy from 70.18% to 99.80%[5]. The 5-stacked model (KNN, SVM_RBF, DT, RF, MLP) achieved perfect scores, with 100% accuracy, MCC, and F1 score, outperforming other models in detecting DDoS attacks in Software-Defined Networks (SDNs) using the Random Forest classifier, with features selected by the Recursive Feature Elimination (RFE) method, achieved 99.97% accuracy, highlighting the effectiveness of 5-stacked model (KNN, SVM_RBF, DT, RF, MLP)[6].

Endnotes

1. M. Mikac and M. Horvatić. "An Approach For Teaching And Understanding Computer Networks Using Realistic Emulation Tool." In ICERI2019 Proceedings, pp. 1209--1219, 2019
2. J. Fox and M. Bouchet-Vala. Using the R Commander: A Point-and-Click Interface for R. Chapman and Hall/CRC Press: Boca Raton FL, 2017
3. J. Ye, X. Cheng, J. Zhu, L. Feng, L. Song. A DDoS attack detection method based on SVM in software defined network. Security and Communication Networks, Vol. 2018, No. 1 Wiley Online Library p. 9804061, 2018
4. Z. Ma, B. Li. A DDoS attack detection method based on SVM and K-nearest neighbour in SDN environment. International Journal of Computational Science and Engineering , Vol. 23, No. 3 Inderscience Publishers (IEL) p. 224-234, 2020
5. M. S. Alkhasawneh. Software defect prediction through neural network and feature selections. Applied Computational Intelligence and Soft Computing, Vol. 2022, No. 1 Wiley Online Library p. 2581832, 2022
6. M. W. Nadeem, H. G. Goh, V. Ponnusamy, Y. Aun. DDoS Detection in SDN using Machine Learning Techniques. Computers, Materials & Continua , Vol. 71, No. 1, 2022

Chapter Five

Conclusion

This chapter offers recommendations for addressing the identified problem, summarizes the study's analysis and interpretation, and suggests directions for future research. Moreover, it discusses the research's implications and contributions to the knowledge base on computer network constraints, computer network datasets, and the application of machine learning for developing models to enhance computer network performance.

5.1 Summary of Results

The first study investigates the impact of different networking approaches on computer network performance. It reveals that a hybrid TCN with SDN exhibits lower bandwidth usage but higher latency compared to traditional networks. Throughput and jitter are influenced by packet size and network type, with non-homogeneous networks potentially increasing jitter. The results underscore the trade-offs between networking approaches and emphasize the need to consider network type, packet size, and application characteristics for optimal performance. The second study focuses on the effects of Background-Traffic and Bandwidth-Limits on SDN performance, demonstrating that higher UDP bandwidth corresponds to increased TCP effects. ANOVA results confirm the significance of these factors on TCP bandwidth, throughput, and latency. The third study explores data flow features in an SDN environment, revealing strong correlations between the number of packets and bytes. The final study evaluates machine learning models for SDN performance under different traffic and bandwidth conditions, highlighting the sensitivity of model performance to these variables. The 5-StkMdl proves superior, achieving high Accuracy, MCC, and F1 scores across various scenarios.

5.2 Recommendations

The study delves into the nuanced dynamics of networking approaches, specifically focusing on traditional, hybrid, and software-defined networking (SDN) environments. The findings underscore significant correlations between packet flow and application performance metrics such as bandwidth usage, throughput, latency, and jitter. Notably, the hybrid TCN with SDN exhibited the lowest bandwidth usage, while traditional networks showed higher throughput but at the cost of increased latency. The introduction of SDN brought forth notable influences on latency, throughput, and jitter, particularly under varying UDP Bandwidth-Limits. The study further scrutinizes data flow features in an SDN context, revealing strong correlations between the number of packets and bytes in the flow, with implications for performance optimization. Machine learning models, particularly the 5-StkMdl, emerged as powerful tools for predicting SDN performance under different traffic and bandwidth conditions, emphasizing the intricate relationship between model sensitivity, traffic patterns, and Bandwidth-Limitations. Overall, these findings provide valuable insights into the trade-offs and considerations necessary for optimizing computer network performance in diverse and evolving environments. The study delves into the impact of different networking approaches on computer network performance, focusing on bandwidth usage, throughput, latency, and jitter. Findings indicate that a hybrid TCN with SDN exhibits lower bandwidth usage but may lead to increased latency, emphasizing trade-offs between networking approaches. Another investigation into SDN explores the influence of Background-Traffic and Bandwidth-Limits on latency, throughput, jitter, and datagram loss. Results show higher UDP Bandwidth-Limits correlate with increased TCP effects, with nuanced effects on latency. The study extends to data flow features in an SDN environment, revealing the influence of traffic type and Bandwidth-Limits on switches, packet flow duration, and other parameters. Correlation analyses underscore strong relationships between the number of packets and bytes in the flow. Additionally, a stack machine learning model evaluation showcases varying performance across traffic conditions and Bandwidth-Limits, with SVM_RBF consistently outperforming, and MLP underperforming,

highlighting the model's sensitivity to these variables. Finally, combined machine learning models demonstrate that the 5-StkMdl excels, achieving high accuracy, MCC, and F1 scores across diverse treatments, showcasing its robust performance under different conditions. These findings collectively contribute valuable insights into optimizing computer network performance in diverse scenarios.

5.3 Contribution to Knowledge

The first study significantly contributes to our understanding of computer network performance by investigating the impact of different networking approaches on key metrics such as bandwidth, throughput, latency, and jitter. The findings reveal nuanced relationships between network types, packet sizes, and applications. The study underscores the trade-offs associated with various networking strategies and highlights the importance of considering network type, packet size, and pplication characteristics in optimizing computer network performance. Additionally, the statistical analyses, including ANOVA and Duncan's Multiple Range Test, provide a robust framework for assessing the significance of different factors.

In the second study, the examination of Background-Traffic and Bandwidth-Limits on the performance metrics of software-defined networking (SDN) adds valuable insights to the field. The study unveils complex relationships between UDP Bandwidth-Limits and TCP bandwidth effects, particularly during specific tasks like email downloads and web browsing. The significance of Background-Traffic and Bandwidth-Limits on TCP bandwidth, throughput, and latency is confirmed through ANOVA results. Furthermore, Duncan's multiple range test offers a detailed understanding of the specific effects of different traffic types and Bandwidth-Limits on various performance metrics, providing a comprehensive view of their individual and interactive impacts on SDN performance.

The third study delves into data flow features in an SDN environment across diverse traffic and

bandwidth conditions. The comprehensive dataset and subsequent analyses reveal the intricate relationships between dataflow features under specific conditions. The study contributes to our understanding of how traffic type and Bandwidth-Limit influence data flow characteristics in an SDN environment. The consistent strong correlation between the number of packets and bytes in the flow suggests potential intrinsic relationships. These findings offer valuable insights into how different traffic and bandwidth conditions impact the performance and characteristics of data flow in SDN, contributing to the optimization of SDN environments.

Lastly, the evaluation of a stack machine learning model for different datasets reflecting the influence of traffic and bandwidth on SDN performance is a significant contribution. The study not only tests various models and stacking configurations but also considers the sensitivity of the model's performance to different traffic conditions and Bandwidth-Limits. The detailed evaluation of different models under various conditions provides valuable guidance for choosing appropriate models in SDN environments. The results underscore the importance of considering both traffic conditions and Bandwidth-Limits in optimizing machine learning models for SDN performance. Overall, these studies collectively contribute to the evolving body of knowledge in the field of computer networks and SDN.

5.4 Suggestions for Further Studies

The research on the impact of different networking approaches offers valuable insights into optimizing computer network performance. The study establishes that the choice of networking approach significantly influences bandwidth usage, throughput, latency, and jitter. Notably, a hybrid TCN with SDN showed promise in minimizing bandwidth usage, while traditional networks exhibited higher throughput. Latency and jitter were influenced by the combination of traditional and SDN networks. The findings underscore the importance of considering network type, packet size, and application characteristics when designing and optimizing computer networks. Future

studies could delve deeper into the specific configurations and protocols within SDN that contribute to latency, bandwidth, and jitter variations.

The investigation into the impact of Background-Traffic and Bandwidth-Limits on SDN performance illuminates critical relationships. Notably, higher UDP Bandwidth-Limits affecting TCP bandwidth, throughput, and latency during activities such as email downloads and web browsing are identified. The ANOVA results and Duncan's multiple range test contribute statistical rigor, detailing the significance of Background-Traffic types and Bandwidth-Limits on various performance metrics. This study lays the foundation for more nuanced exploration, such as nderstanding the dynamic nature of Background-Traffic and its impact on specific SDN applications. The exploration of data flow features in an SDN environment provides a granular understanding of how traffic type and Bandwidth-Limit influence switches, packet flow duration, and other parameters. The consistent strong correlation between the number of packets and bytes suggests intrinsic relationships that warrant further investigation. This study's focus on diverse traffic and bandwidth conditions opens avenues for future research to delve into the specifics of how certain conditions impact data flow characteristics. Additionally, applying machine learning techniques to predict or optimize data flow features under varying circumstances could be a promising direction for further inquiry.

The evaluation of stack machine learning models in the context of SDN presents a nuanced erspective on the sensitivity of model performance to traffic conditions and Bandwidth-Limits. The consistent outperformance of SVM_RBF and the lower metrics associated with MLP under 5Stacked suggest potential patterns that merit deeper exploration. Future studies could investigate the interpretability of these models, providing insights into the specific features that contribute to their varying performances under different traffic and bandwidth scenarios. Furthermore, understanding how these models generalize across diverse SDN environments and real-world network configurations could enhance the applicability of the findings.

The examination of integrated machine learning models sheds light on their efficacy in predicting SDN performance under varying conditions. The superior performance of the 5-StkMdl across multiple metrics and datasets indicates its robustness. However, the observed differences in performance based on dataset sizes and configurations suggest the need for further investigation into model stability and scalability. Future research could focus on refining these models, exploring additional features, and considering real-world deployment challenges to enhance the practical utility of such predictive models in SDN environments.

Bibliography

Book

Fox, J.: 2017, Using the R Commander: A Point-and-Click Interface for R, Chapman and Hall/CRC Press, Boca Raton FL. URL: http://socserv.mcmaster.ca/jfox/Books/RCommander/

Fox, J. and Bouchet-Valat, M.: 2018, Rcmdr: R Commander. URL: http://socserv.socsci.mcmaster.ca/jfox/Misc/Rcmdr/

Horner, L. J., Tutschku, K., Fumagalli, A. and Ramanathan, S.: 2023, Virtualizing 5G and Beyond 5G Mobile Network, Artech House.

Melin, P., Miramontes, I. and Arechiga, G. P.: 2022, Nature-inspired optimization of type-2 fuzzy neural hybrid models for classification in medical diagnosis, SpringerBriefs Appl. Sci. Technol., Cham: Springer.

Raza, K. (ed.): 2022, Computational intelligence in oncology. Applications in diagnosis, prognosis and therapeutics of cancers, Vol. 1016 of Stud. Comput. Intell., Singapore: Springer.

Thakur, S. and Jha, S. K.: 2023, Cloud computing and its emerging trends on big data analytics, 2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC), IEEE, pp. 1159–1164.

Zhou, Z.-H.: 2021, Ensemble Learning, Springer Singapore, Singapore, pp. 181–210. URL: https://doi.org/10.1007/978-981-15-1967-38

Conference Paper

Abar, T., Letaifa, A. B. and Asmi, S. E.: 2019, Real time anomaly detection-based qoe feature selection and ensemble learning for http video services, IEEE, Hammamet, Tunisia, pp. 1–6.

Alghamdi, A., Paul, D. J. and Sadgrove, E. J.: 2021, A restful northbound interface for applications in software defined networks., WEBIST, pp. 453–459.

Alibrahim, H. and Ludwig, S. A.: 2021, Hyperparameter optimization: Comparing genetic

algorithm against grid search and bayesian optimization, 2021 IEEE Congress on Evolutionary Computation (CEC), IEEE, pp. 1551–1559.

Amarudin, Ferdiana, R. and Widyawan: 2022, New approach of ensemble method to improve performance of ids using s-sdn classifier, IEEE, Solo, Indonesia, pp. 463–468.

Anifowose, F., Ayadiuno, C. and Reshedan, F.: 2019, Feature selection based hybrid machine learning approach to formation cementation factor prediction, SPE Kuwait Oil and Gas Show and Conference, SPE, p. D033S017R002.

Arun, V., Alizadeh, M. and Balakrishnan, H.: 2022, Starvation in end-to-end congestion control, Proceedings of the ACM SIGCOMM 2022 Conference, pp. 177–192.

Christila, S. A. and Sivakumar, R.: 2022, Multi-layer ensemble deep reinforcement learning based ddos attack detection and mitigation in cloud-sdn environment, IEEE, Bangalore, India, pp. 451–455.

Comaneci, D. and Dobre, C.: 2018, Securing networks using SDN and machine learning, 2018 IEEE International Conference on Computational Science and Engineering (CSE), IEEE.

Deepa, V., Sudar, K. M. and Deepalakshmi, P.: 2019, Design of ensemble learning methods for ddos detection in sdn environment, IEEE, Vellore, India, pp. 1–6.

Dey, R. and Mathur, R.: 2023, Ensemble learning method using stacking with base learner, a comparison, International Conference on Data Analytics and Insights, Springer, pp. 159–169.

Eom, W.-J., Song, Y.-J., Park, C.-H., Kim, J.-K., Kim, G.-H. and Cho, Y.-Z.: 2021, Network traffic classification using ensemble learning in software-defined networks, IEEE, Jeju Island, Korea (South), pp. 089–092.

Fan, Z. and Liu, R.: 2017, Investigation of machine learning based network traffic classification, 2017 International Symposium on Wireless Communication Systems (ISWCS) pp. 1–6.
URL: https://api.semanticscholar.org/CorpusID:20913098

Fancy, C. and Pushpalatha, M.: 2017, Performance evaluation of sdn controllers pox and floodlight in mininet emulation environment, Proceedings of the International Conference on Intelligent Sustainable Systems (ICISS 2017).

Feurer, M. and Hutter, F.: 2019, Hyperparameter optimization, Automated machine learning, Springer, Cham, pp. 3–33.

Grina, F., Elouedi, Z. and Lefèvre, E.: 2021, Uncertainty-aware resampling method for imbalanced classification using evidence theory, Symbolic and quantitative approaches to reasoning with uncertainty. 16th European conference, ECSQARU 2021,

Prague, Czech Republic, September 21–24, 2021. Proceedings, Cham: Springer, pp. 342–353.

Hartmann, T., Moawad, A., Schockaert, C., Fouquet, F. and Le Traon, Y.: 2019, Meta-modelling meta-learning, 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, pp. 300–305.

Houda, Z. A. E., Brik, B. and Khoukhi, L.: 2022, Ensemble learning for intrusion detection in sdn-ased zero touch smart grid systems, IEEE, Edmonton, AB, Canada, pp. 149–156.

Jadin, M., Tilmans, O., Mawait, M. and Bonaventure, O.: 2020, Educational virtual routing labs with ipmininet, ACM SIGCOMM Education Workshop, pp. 1–5.

Kumar, G., Dukkipati, N., Jang, K., Wassel, H. M., Wu, X., Montazeri, B., Wang, Y., Springborn, K., Alfeld, C. and Ryan, M.: 2020, Swift: Delay is simple and effective for congestion control in the datacenter, Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, pp. 514–528.

Lee, K., Laskin, M., Srinivas, A. and Abbeel, P.: 2021, Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning, International Conference on Machine Learning, PMLR, pp. 6131–6141.

Li, Y., Guo, X., Pang, X., Peng, B., Li, X. and Zhang, P.: 2020, Performance analysis of floodlight

and ryu sdn controllers under mininet simulator, IEEE/CIC International Conference on Communications in China (ICCC Workshops).

Lin, Z. and Hongle, D.: 2020, Research on sdn intrusion detection based on online ensemble learning algorithm, IEEE, Haikou City, China, pp. 114–118.

Liu, F., Kibalya, G., Santhosh Kumar, S. and Zhang, P.: 2021, Challenges of traditional networks and development of programmable networks, Software defined internet of everything, Springer, pp. 37–61.

Liu, L., Essam, D. and Lynar, T.: 2021, On quantifying the complexity of iot traffic, 2021 IEEE 46th Conference on Local Computer Networks (LCN), IEEE, pp. 379–382.

Mikac, M. and Horvatić, M.: 2019, An approach for teaching and understanding computer networks using realistic emulation tool, ICERI2019 Proceedings, IATED, pp. 1209–1219.

Mohammed, A. R., Mohammed, S. A. and Shirmohammadi, S.: 2019, Machine learning and deep learning based traffic classification and prediction in software defined.

Mostacero-Agama, L. and Shiguihara, P.: 2022, Analysis of internet service latency and its impact on internet of things (iot) applications, 2022 IEEE Engineering International Research Conference (EIRCON), IEEE, pp. 1–4.

Mukherjee, B. K., Pappu, S. I., Islam, M. J. and Acharjee, U. K.: 2020, An sdn based distributed iot network with nfv implementation for smart cities, Cyber Security and Computer Science: Second EAI International Conference, ICONCS 2020, Dhaka, Bangladesh, February 15-16, 2020, Proceedings 2, Springer, pp. 539–552.

Pakzad, F., Portmann, M., Tan, W. L. and Indulska, J.: 2014, Efficient topology discovery in software defined networks, 2014 8th international conference on signal processing and communication systems (ICSPCS), IEEE, pp. 1–8.

Raghunath, K. and Krishnan, P.: 2018, Towards a secure sdn architecture, 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT),

IEEE, pp. 1–7.

Rajawat, S. S., Khatri, P. and Surange, G.: 2022, Sniffit: A packet sniffing tool using wireshark, International Conference on Communication, Networks and Computing, Springer, pp. 203–212.

Roque, A. S., Jazdi, N., Freitas, E. P. and Pereira, C. E.: 2020, Performance analysis of in-vehicle distributed control systems applying a real-time jitter monitor, 2020 IEEE 18th International Conference on Industrial Informatics (INDIN), Vol. 1, IEEE, pp. 663–668.

Sebopetse, N. S., Burger, C. R., Mofolo, M. and Lysko, A. A.: 2021, Measuring with jperf and psping: Throughput and estimated packet delivery delay vs tcp window size & parallel streams, 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), Vol. 1, IEEE, pp. 828–832.

Shah, G., Valiente, R., Gupta, N., Gani, S. O., Toghi, B., Fallah, Y. P. and Gupta, S. D.: 2019, Real-time hardware-in-the-loop emulation framework for dsrc-based connected vehicle applications, 2019 IEEE 2nd Connected and Automated Vehicles Symposium (CAVS), IEEE, pp. 1–6.

Sharma, K. K. and Sood, M.: 2015, Mininet as a container based emulator for software d fined networks. Shirmarz, A., Ghaffari, A., Mohammadi, R. and Akleylek, S.: 2021, Ddos attack detection accuracy improvement in software defined network (sdn) using ensemble classification, IEEE, Ankara, Turkey, pp. 111–115.

Sturzinger, E. and Cilenti, S.: 2019, A hybrid software defined network platform for undergraduate research and education.proceedings of the national conference on undergraduate research (ncur) kennesaw state university, kennesaw, georgia , april 11-13, Proceedings of The National Conference On Undergraduate Research (NCUR) Kennesaw State University, Kennesaw, Georgia , April 11-13.

Todorov, N., Ganchev, I. and OâDroma, M.: 2021, Internet performance profiling of countries,

Advances in Computing and Network Communications: Proceedings of CoCoNet 2020, Volume 2, Springer, pp. 503–518.

Trochun, Y., Stirenko, S., Rokovyi, O., Alienin, O., Pavlov, E. and Gordienko, Y.: 2021, Hybrid classic-quantum neural networks for image classification.

Villalobos-Arias, L., Quesada-López, C., Guevara-Coto, J., Martínez, A. and Jenkins, M.: 2020, Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation, Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, ACM.

Wang, D. and Yue, X.: 2019, The weighted multiple meta-models stacking method for regression problem, 2019 Chinese Control Conference (CCC), IEEE, pp. 7511–7516.

Wang, Y., Chen, C. and Huang, W.: 2021, Design of quantum filter for hybrid quantum-classical convolutional neural networks.

Xiao, Y., Varvello, M. and Kuzmanovic, A.: 2022, Monetizing spare bandwidth: The case of distributed vpns, Proceedings of the ACM on Measurement and Analysis of Computing Systems 6(2), 1–27.

Yang, G., Jin, H., Kang, M., Moon, G. J. and Yoo, C.: 2020, Network monitoring for sdn virtual networks, IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, pp. 1261–1270.

Yang, Z., Cui, Y., Li, B., Liu, Y. and Xu, Y.: 2019, Software-defined wide area network (sd-wan): Architecture, advances and opportunities, 2019 28th International Conference on Computer Communication and Networks (ICCCN), IEEE, pp. 1–9.

Yuan, X., Liu, Z., Park, Y., Hu, H. and Li, H.: 2020, Teaching sdn security using hands-on labs in cloudlab, Journal of The Colloquium for Information Systems Security Education, Vol. 7, pp. 6–6.

Zhao, X., Jahre, M., Tang, Y., Zhang, G. and Eeckhout, L.: 2023, Nuba: Non-uniform bandwidth .

Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASP-LOS 2023, Association for Computing Machinery, New York, NY, USA, p. 544â559.
URL: https://doi.org/10.1145/3575693.3575745

Zulu, L. L., Ogudo, K. A. and Umenne, P. O.: 2018, Emulating software defined network using mininet and opendaylight controller hosted on amazon web services cloud platform to demonstrate a realistic programmable network, 2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC), IEEE, pp. 1–7.

Electronic Sources

Arahunashi, A. K., Neethu, S. and Aradhya, H. V. R.: 2019, Performance analysis of various sdn controllers in mininet emulator.

Barrett, R., Facey, A., Nxumalo, W., Rogers, J., Vatcher, P. and St-Hilaire, M.: 2017, Dynamic traffic diversion in sdn: testbed vs mininet.

Chaurasia, A., Mishra, S. N. and Chinara, S.: 2019, Performance evaluation of software-defined wireless networks in it-sdn and mininet-wifi.

Fancy, C. and Pushpalatha, M.: 2017, Performance evaluation of sdn controllers pox and floodlight in mininet emulation environment.

Flauzac, O., Robledo, E. M. G. and Nolot, F.: 2019, Is mininet the right solution for an sdn testbed?

Hasan, M., Dahshan, H., Abdelwanees, E. and Elmoghazy, A.: 2020, Sdn mininet emulator benchmarking and result analysis.

Jawaharan, R., Mohan, P. M., Das, T. and Gurusamy, M.: 2018, Empirical evaluation of sdn controllers using mininet/wireshark and comparison with cbench.

Jitnukulsiri, S. and Aswakul, C.: 2019, Validation of sdn emulator based on mininet and onos controller for iec 61850 packet delay measurement.

Lena, G. D., Tomassilli, A., Saucez, D., Giroire, F., Turletti, T. and Lac, C.: 2019, Mininet on steroids: exploiting the cloud for mininet performance.

Li, Y., Guo, X., Pang, X., Peng, B., Li, X. and Zhang, P.: 2020, Performance analysis of floodlight and ryu sdn controllers under mininet simulator.

Meeker, M. and Wu, L.: 2018, Internet trends 2018.

Rathi, V. K. and Singh, K.: 2018, Sdn layer 2 switch simulation using mininet and opendaylight.

Journal

Alkhasawneh, M. S.:2022, Software defect prediction through neural network and feature selections, Applied Computational Intelligence and Soft Computing 2022(1), 2581832.

Almadani, B., Beg, A. and Mahmoud, A.: 2021, Dsf: A distributed sdn control plane framework for the east/west interface, IEEE Access 9, 26735–26754.

Alzubi, A.: 2022, Learner performance prediction in the e-learning platform using the optimized deep long short-term memory classifier, International Journal of Wavelets, Multiresolution and Information Processing 20(2), 29. Id/No 2150051.

Arahunashi, A. K., Neethu, S. and Aradhya, H. V. R.: 2019, Performance analysis of various sdn controllers in mininet emulator, IEEE Xplore .

Arthur, D.: 2022, A hybrid quantum-classical neural network architecture for binary classification, arXiv preprint arXiv:2201.01820 .

Babu, G. J., Banks, D., Cho, H., Han, D., Sang, H. and Wang, S.: 2021, A statistician teaches deep learning, Journal of Statistical Theory and Practice 15, 1–23.

Bakhteev, O. Y. and Strijov, V. V.: 2017, Comprehensive analysis of gradient-based hyperparameter optimization algorithms, Annals of Operations Research 289(1), 51–65.

Bao, K., Matyjas, J. D., Hu, F. and Kumar, S.: 2018, Intelligent software-defined mesh networks with link-failure adaptive traffic balancing, IEEE Transactions on Cognitive

Communications and Networking 4(2), 266–276.

Bhatia, J. and Patel, P.: 2015, Mininet–an emulator for prototyping large network topologies on a single machine, Admin Insight pp. 51–56.
URL: www.OpensourceForU.com

Buhendwa, A. B., Bezgin, D. A. and Adams, N. A.: 2022, Consistent and symmetry preserving data-driven interface reconstruction for the level-set method, Journal of Computational Physics 457, 18. Id/No 111049.

Caigny, A. D., Coussement, K. and Bock, K. W. D.: 2018, A new hybrid classification algorithm for customer churn prediction based on logistic regression and decision trees, European Journal of Operational Research 269(2), 760–772.

Charoenkwan, P., Schaduangrat, N., Nantasenamat, C., Piacham, T. and Shoombuatong, W.: 2019, iQSP: A sequence-based tool for the prediction and analysis of quorum sensing peptides using informative physicochemical properties, International Journal of Molecular Sciences 21(1), 75.

Chaudhari, J., Kirange, D., Bhagat, K. and Patil, S.: 2019, Ealuation of bandwidth utilization in sdn, Journal of Xiâan Shiyou University ISSN No 1673, 064X.

Côme, E., Jouvin, N., Latouche, P. and Bouveyron, C.: 2021, Hierarchical clustering with discrete latent variable models and the integrated classification likelihood, Advances in Data Analysis and Classification. ADAC 15(4), 957–986.

Coscrato, V., de Almeida Inacio, M. H. and Izbicki, R.: 2020, The nn-stacking: Feature weighted linear stacking through neural networks, Neurocomputing 399, 141–152.

Dong, X., Yu, Z., Cao, W., Shi, Y. and Ma, Q.: 2020, A survey on ensemble learning, Frontiers of Computer Science 14, 241–258.

Dudi, B. and Rajesh, V.: 2022, Optimized threshold-based convolutional neural network for plant leaf classification: a challenge towards untrained data, Journal of Combinatorial

Optimization 43(2), 312–349.

Estrada-Solano, F., Ordonez, A., Granville, L. Z. and Rendon, O. M. C.: 2017, A framework for sdn integrated management based on a cim model and a vertical management plane, Computer Communications 102, 150–164.

Fang, Z., Wang, Y., Peng, L. and Hong, H.: 2020, A comparative study of heterogeneous ensemble-learning techniques for landslide susceptibility mapping, International Journal of Geographical Information Science 35(2), 321–347.

Fraihat, A.: 2021, Computer networking layers based on the osi model, Test Eng. Manag 83, 6485–6495.

Garg, R.: 2018, A primer to ensemble learning–bagging and boosting, Analytics India Magazine .

Gomes, H. M., Read, J., Bifet, A., Barddal, J. P. and Gama, J.: 2019, Machine learning for streaming data: state of the art, challenges, and opportunities, ACM SIGKDD Explorations Newsletter 21(2), 6–22.

Guan, W., Zhang, H. and Leung, V. C.: 2020, Analysis of traffic performance on network slicing using complex network theory, IEEE Transactions on vehicular technology 69(12), 15188–15199.

Guevara, C. and Santos, M.: 2021, Intelligent models for movement detection and physical evolution of patients with hip surgery, Logic Journal of the IGPL 29(6), 874–888.

Guo, Y., Hu, G. and Shao, D.: 2022, Qogmp: Qos-oriented global multi-path traffic scheduling algorithm in software defined network, Scientific Reports 12(1), 1–12.

URL: https://doi.org/10.1038/s41598-022-18919-w

Haile, H., Grinnemo, K.-J., Ferlin, S., Hurtig, P. and Brunstrom, A.: 2021, End-to-end congestion control approaches for high throughput and low delay in 4g/5g cellular networks, Computer Networks 186, 107692.

Hardesty, J. L.: 2016, Transitioning from XML to RDF: Considerations for an effective move

towards Linked Data and the Semantic Web, Information Technology and Libraries (Online) 35(1), 51.

Hassan, W., Chou, T.-S., Li, X., Appiah-Kubi, P. and Tamer, O.: 2019, Latest trends, challenges and solutions in security in the era of cloud computing and software defined networks, Int J Inf & Commun Technol ISSN 2252(8776), 8776.

Hosseinpour, M., Ghaemi, S., Khanmohammadi, S. and Daneshvar, S.: 2022, A hybrid highâorder typeâ2 fcm improved random forest classification method for breast cancer risk assessment, Applied Mathematics and Computation 424, 127038.
URL: https://www.sciencedirect.com/science/article/pii/S0096300322001242

Isolani, P. H., Kulenkamp, D. J., Marquez-Barja, J. M., Granville, L. Z., LatrÃ©, S. And Syrotiuk, V. R.: 2021, Support for 5g mission-critical applications in software-defined ieee 802.11 networks, Sensors 21(693), 693.
URL: https://www.mdpi.com/1424-8220/21/3/693

Jameel, M. A., Kanakis, T., Turner, S., Al-Sherbaz, A. and Bhaya, W. S.: 2022, A re-inforcement learning-based routing for real-time multimedia traffic transmission over software-defined networking, Electronics 11(2441), 2441.
URL: https://www.mdpi.com/2079-9292/11/15/2441

Kanazawa, T. and Wettig, T.: 2017, Complete random matrix classification of syk models with ð=0, 1 and 2 supersymmetry, JHEP 09, 050.

Kansal, M., Singh, P., Singh, M. K. and Varshney, S.: 2023, A systematic study of services and security model in cloud computing: A brief overview, Convergence of Cloud Computing, AI, and Agricultural Science pp. 1–16.

Kaur, C.: 2022, Incorporating sentimental analysis into development of a hybrid classification model: A comprehensive study, International Journal of Health Sciences 6, 1709–1720.

Khairi, M. H., Ariffin, S. H., Latiff, N. A. and Yusof, K. M.: 2021, Generation and collection of

data for normal and conflicting flows in software defined network flow table, Indonesian J. Electr. Eng. Comput. Sci, 22(1), 307.

Lena, G. D., Tomassilli, A., Saucez, D., Giroire, F., Turletti, T. and Lac, C.: 2019, Mininet on steroids: Exploiting the cloud for mininet performance, IEEE Xplore .

Lu, Y., Zhao, G., Chakraborty, C., Xu, C., Yang, L. and Yu, K.: 2023, Time sensitive networking-driven deterministic low-latency communication for real-time telemedicine and e-health services, IEEE Transactions on Consumer Electronics .

Ma, Z. and Li, B.: 2020, A ddos attack detection method based on svm and k-nearest neighbour in sdn environment, International Journal of Computational Science and Engineering 23(3), 224–234.

Marino, M. F. and Pandolfi, S.: 2022, Hybrid maximum likelihood inference for stochastic block models, Computational Statistics and Data Analysis 171, 19. Id/No 107449.

Matloob, F., Ghazal, T. M., Taleb, N., Aftab, S., Ahmad, M., Khan, M. A., Abbas, S. and Soomro, T. R.: 2021, Software defect prediction using ensemble learning: A systematic literature review, IEEE Access 9, 98754–98771.

Mazhar, T., Malik, M. A., Mohsan, S. A. H., Li, Y., Haq, I., Ghorashi, S., Karim, F. K. And Mostafa, S. M.: 2023, Quality of service (qos) performance analysis in a traffic engineering model for next-generation wireless sensor networks, Symmetry 15(2), 513.

Miguel-Alonso, J.: 2022, A research review of openflow for datacenter networking, IEEE Access .

Mishra, T. K., Sahoo, K. S., Bilal, M., Shah, S. C. and Mishra, M. K.: 2023, Adaptive congestion control mechanism to enhance tcp performance in cooperative iov, IEEE Access 11, 9000–9013.

Nadeem, M. W., Goh, H. G., Ponnusamy, V. and Aun, Y.: 2022, Ddos detection in sdn using machine learning techniques., Computers, Materials & Continua 71(1).

Nikandish, G., Staszewski, R. B. and Zhu, A.: 2020, Breaking the bandwidth limit: A review of

broadband doherty power amplifier design for 5g, IEEE Microwave Magazine 21(4), 57–75.

Njah, Y., Pham, C. and Cheriet, M.: 2020, Service and resource aware flow management scheme for an sdn-based smart digital campus environment, IEEE Access 8, 119635–119653. URL: https://ieeexplore.ieee.org/document/9127419/

Nkenyereye, L., Nkenyereye, L., Pham, Q.-V. and Song, J.: 2021, Efficient RSU selection scheme for fog-based vehicular software-defined network, IEEE Transactions on Vehicular Technology pp. 1–1.

Nti, I. K., Adekoya, A. F. and Weyori, B. A.: 2020, A comprehensive evaluation of ensemble learning for stock-market prediction, Journal of Big Data 7(1), 1–40.

Omeka, M. E., Igwe, O., Onwuka, O. S., Nwodo, O. M., Ugar, S. I., Undiandeye, P. A. and Anyanwu, I. E.: 2023, Efficacy of gis-based ahp and data-driven intelligent machine learning algorithms for irrigation water quality prediction in an agricultural-mine district within the lower benue trough, nigeria, Environmental Science and Pollution Research pp. 1–30.

Orozco-Santos, F., Sempere-Paya, V., Silvestre-Blanes, J. and Vera-Perez, J.: 2022, Scalability enhancement on software defined industrial wireless sensor networks over tsch, IEEE Access 10, 107137–107151. URL: https://ieeexplore.ieee.org/document/9913479/

Poularakis, K., Tassiulas, L. and Lakshman, T.: 2021, Modeling and optimization in software-defined networks, Synthesis Lectures on Learning, Networks, and Algorithms 2(2), 1–174. URL: https://doi.org/10.2200/S01099ED1V01Y202105LNA027

Rai, A., D Vyavahare, P. and Jain, A.: 2019, Distributed dos attack detection and mitigation in software defined network (sdn), Proceedings of Recent Advances in Interdisciplinary Trends in Engineering & Applications (RAITEA) .

Rasool, R. U., Ashraf, U., Ahmed, K., Wang, H., Rafique, W. and Anwar, Z.: 2019, Cyber-pulse: A machine learning based link flooding attack mitigation system for software defined

networks, IEEE Access 7, 34885–34899.

Sadiku, I. B., Ajayi, W., Sakpere, W., John-Dewole, T. and Badru, R.: 2022, Effect of traditional and software-defined networking on performance of computer network, Scientific Journal of Informatics 9(2), 111–122.

Sagingalieva, A., Kurkin, A., Melnikov, A., Kuhmistrov, D., Perelshtein, M., Melnikov, A., Skolik, A. and Von Dollen, D.: 2022, Hyperparameter optimization of hybrid quantum neural networks for car classification.

Sayjari, T., Silveira, R. M. and Margi, C. B.: 2023, Application-aware scheduling for ieee 802.15.4e time-slotted channel hopping using software-defined wireless sensor network slicing, Sensors 23(7143), 7143. URL: https://www.mdpi.com/1424-8220/23/16/7143

Schetakis, N., Aghamalyan, D., Griffin, P. and Boguslavsky, M.: 2022, Review of some existing qml frameworks and novel hybrid classicalâquantum neural networks realising binary classification for the noisy datasets, Sci. Rep. 12(1), 11927.

Shahabi, H., Jarihani, B., Tavakkoli Piralilou, S., Chittleborough, D., Avand, M. and Ghorbanzadeh, O.: 2019, A semi-automated object-based gully networks detection using different machine learning models: a case study of bowen catchment, queensland, australia, Sensors 19(22), 4893.

Shamsudin, H., Sabudin, M. and Yusof, U. K.: 2019, Hybridisation of rf (xgb) to improve the tree-based algorithms in learning style prediction, IAES International Journal of Artificial Intelligence 8(4), 422.

Tian, J.-x. and Zhang, J.: 2022, Breast cancer diagnosis using feature extraction and boosted C5.0 decision tree algorithm with penalty factor, Mathematical Biosciences and Engineering 19(3), 2193–2205.

Yang, P., Xiong, N. and Ren, J.: 2020, Data security and privacy protection for cloud storage: A survey, IEEE Access 8, 131723–131740.

Ye, J., Cheng, X., Zhu, J., Feng, L. and Song, L.: 2018, A ddos attack detection method based on svm in software defined network, Security and Communication Networks 2018(1), 9804061.

Yusuf, M. N., bin Abu Bakar, K., Isyaku, B., Osman, A. H., Nasser, M. and Elhaj, F. A.: 2023, Adaptive path selection algorithm with flow classification for software-defined networks, Mathematics 11(1404), 1404. URL: https://www.mdpi.com/2227-7390/11/6/1404

Zabian, A. and Ibrahim, A. Z.: 2022, Hybrid mathematical model for data classification and prediction: case study COVID-19, International Journal of Mathematics and Computer Science 17(3), 995–1006. URL: ijmcs.future-in-tech.net/17.3/R-Arwa.pdf

Zhang, Y., Liu, J. and Shen, W.: 2022, A review of ensemble learning algorithms used in remote sensing applications, Applied Sciences 12(17), 8654.

Zhao, H., Chen, Q., Qiu, Y., Wu, M., Shen, Y., Leng, J., Li, C. and Guo, M.: 2018, Bandwidth and locality aware task-stealing for manycore architectures with bandwidth-asymmetric memory, p. 1Â26. URL: https://doi.org/10.1145/3291058

Zieliński, B.: 2023, Assessment of iperf as a tool for lan throughput prediction, International Journal of Electronics and Telecommunications pp. 523–528.

Zou, T., Wang, Y. and WU, C.: 2019, Review of network background traffic classification and identification, Journal of Computer Applications 39(3), 802.

Thesis

Alghamdi, K. Y.: 2021, A Software Defined paradigm for mobile networks: A feasible SDN based architecture solution for 5G networks, PhD thesis.

Azzawi, H.: 2019, Computational models and approaches for lung cancer diagnosis, PhD thesis, Deakin University.

Bekoe, S.: 2020, Network traffic analysis using wireshark in solving network problems, PhD thesis, Unversity Of Education, Winneba.

Appendices

Appendix A: Topology in Mininet for LANs - SDN - Internet Experiment

```
#!/usr/bin/env python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
def myNetwork():
    net = Mininet( topo=None,
                   build=False,
                   ipBase='192.168.8.0/24')
    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         protocol='tcp',
                         port=6633)
    info( '*** Add switches\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch)
    s6 = net.addSwitch('s6', cls=OVSKernelSwitch)
    s7 = net.addSwitch('s7', cls=OVSKernelSwitch)
    s8 = net.addSwitch('s8', cls=OVSKernelSwitch)
    s9 = net.addSwitch('s9', cls=OVSKernelSwitch)
    info( '*** Add hosts\n')
#    h1 = net.addHost('h1', cls=Host, ip='192.168.8.1', defaultRoute=None)
#    h2 = net.addHost('h2', cls=Host, ip='192.168.8.2', defaultRoute=None)
#    h3 = net.addHost('h3', cls=Host, ip='192.168.8.3', defaultRoute=None)
#    h4 = net.addHost('h4', cls=Host, ip='192.168.8.4', defaultRoute=None)
#    h5 = net.addHost('h5', cls=Host, ip='192.168.8.5', defaultRoute=None)
#    h6 = net.addHost('h6', cls=Host, ip='192.168.8.6', defaultRoute=None)
#    h7 = net.addHost('h7', cls=Host, ip='192.168.8.7', defaultRoute=None)
#    h8 = net.addHost('h8', cls=Host, ip='192.168.8.8', defaultRoute=None)
#    h9 = net.addHost('h9', cls=Host, ip='192.168.8.9', defaultRoute=None)
#    h10 = net.addHost('h10', cls=Host, ip='192.168.8.10', defaultRoute=None)
#    h11 = net.addHost('h11', cls=Host, ip='192.168.8.11', defaultRoute=None)
#    h12 = net.addHost('h12', cls=Host, ip='192.168.8.12', defaultRoute=None)
    h13 = net.addHost('h13', cls=Host, ip='192.168.8.13', defaultRoute=None)
    h14 = net.addHost('h14', cls=Host, ip='192.168.8.14', defaultRoute=None)
    h15 = net.addHost('h15', cls=Host, ip='192.168.8.15', defaultRoute=None)
    h16 = net.addHost('h16', cls=Host, ip='192.168.8.16', defaultRoute=None)
    h17 = net.addHost('h17', cls=Host, ip='192.168.8.17', defaultRoute=None)
```

```
h18 = net.addHost('h18', cls=Host, ip='192.168.8.18', defaultRoute=None)
h19 = net.addHost('h19', cls=Host, ip='192.168.8.19', defaultRoute=None)
h20 = net.addHost('h20', cls=Host, ip='192.168.8.20', defaultRoute=None)
h21 = net.addHost('h21', cls=Host, ip='192.168.8.21', defaultRoute=None)
h22 = net.addHost('h22', cls=Host, ip='192.168.8.22', defaultRoute=None)
h23 = net.addHost('h23', cls=Host, ip='192.168.8.23', defaultRoute=None)
h24 = net.addHost('h24', cls=Host, ip='192.168.8.24', defaultRoute=None)
h25 = net.addHost('h25', cls=Host, ip='192.168.8.25', defaultRoute=None)
h26 = net.addHost('h26', cls=Host, ip='192.168.8.26', defaultRoute=None)
h27 = net.addHost('h27', cls=Host, ip='192.168.8.27', defaultRoute=None)
h28 = net.addHost('h28', cls=Host, ip='192.168.8.28', defaultRoute=None)
h29 = net.addHost('h29', cls=Host, ip='192.168.8.29', defaultRoute=None)
h30 = net.addHost('h30', cls=Host, ip='192.168.8.30', defaultRoute=None)
h31 = net.addHost('h31', cls=Host, ip='192.168.8.31', defaultRoute=None)
h32 = net.addHost('h32', cls=Host, ip='192.168.8.32', defaultRoute=None)
h33 = net.addHost('h33', cls=Host, ip='192.168.8.33', defaultRoute=None)
h34 = net.addHost('h34', cls=Host, ip='192.168.8.34', defaultRoute=None)
h35 = net.addHost('h35', cls=Host, ip='192.168.8.35', defaultRoute=None)
h36 = net.addHost('h36', cls=Host, ip='192.168.8.36', defaultRoute=None)
h37 = net.addHost('h37', cls=Host, ip='192.168.8.37', defaultRoute=None)
h38 = net.addHost('h38', cls=Host, ip='192.168.8.38', defaultRoute=None)
h39 = net.addHost('h39', cls=Host, ip='192.168.8.39', defaultRoute=None)
h40 = net.addHost('h40', cls=Host, ip='192.168.8.40', defaultRoute=None)
h41 = net.addHost('h41', cls=Host, ip='192.168.8.41', defaultRoute=None)
h42 = net.addHost('h42', cls=Host, ip='192.168.8.42', defaultRoute=None)
h43 = net.addHost('h43', cls=Host, ip='192.168.8.43', defaultRoute=None)
h44 = net.addHost('h44', cls=Host, ip='192.168.8.44', defaultRoute=None)
h45 = net.addHost('h45', cls=Host, ip='192.168.8.45', defaultRoute=None)
h46 = net.addHost('h46', cls=Host, ip='192.168.8.46', defaultRoute=None)
h47 = net.addHost('h47', cls=Host, ip='192.168.8.47', defaultRoute=None)
h48 = net.addHost('h48', cls=Host, ip='192.168.8.48', defaultRoute=None)
h49 = net.addHost('h49', cls=Host, ip='192.168.8.49', defaultRoute=None)
h50 = net.addHost('h50', cls=Host, ip='192.168.8.50', defaultRoute=None)
h51 = net.addHost('h51', cls=Host, ip='192.168.8.51', defaultRoute=None)
h52 = net.addHost('h52', cls=Host, ip='192.168.8.52', defaultRoute=None)
h53 = net.addHost('h53', cls=Host, ip='192.168.8.53', defaultRoute=None)
h54 = net.addHost('h54', cls=Host, ip='192.168.8.54', defaultRoute=None)
h55 = net.addHost('h55', cls=Host, ip='192.168.8.55', defaultRoute=None)
h56 = net.addHost('h56', cls=Host, ip='192.168.8.56', defaultRoute=None)
h57 = net.addHost('h57', cls=Host, ip='192.168.8.57', defaultRoute=None)
h58 = net.addHost('h58', cls=Host, ip='192.168.8.58', defaultRoute=None)
h59 = net.addHost('h59', cls=Host, ip='192.168.8.59', defaultRoute=None)
h60 = net.addHost('h60', cls=Host, ip='192.168.8.60', defaultRoute=None)
h61 = net.addHost('h61', cls=Host, ip='192.168.8.61', defaultRoute=None)
h62 = net.addHost('h62', cls=Host, ip='192.168.8.62', defaultRoute=None)
h63 = net.addHost('h63', cls=Host, ip='192.168.8.63', defaultRoute=None)
h64 = net.addHost('h64', cls=Host, ip='192.168.8.64', defaultRoute=None)
h65 = net.addHost('h65', cls=Host, ip='192.168.8.65', defaultRoute=None)
h66 = net.addHost('h66', cls=Host, ip='192.168.8.66', defaultRoute=None)
h67 = net.addHost('h67', cls=Host, ip='192.168.8.67', defaultRoute=None)
```

```
h68 = net.addHost('h68', cls=Host, ip='192.168.8.68', defaultRoute=None)
h69 = net.addHost('h69', cls=Host, ip='192.168.8.69', defaultRoute=None)
h70 = net.addHost('h70', cls=Host, ip='192.168.8.70', defaultRoute=None)
h71 = net.addHost('h71', cls=Host, ip='192.168.8.71', defaultRoute=None)
h72 = net.addHost('h72', cls=Host, ip='192.168.8.72', defaultRoute=None)
h73 = net.addHost('h73', cls=Host, ip='192.168.8.73', defaultRoute=None)
h74 = net.addHost('h74', cls=Host, ip='192.168.8.74', defaultRoute=None)
h75 = net.addHost('h75', cls=Host, ip='192.168.8.75', defaultRoute=None)
h76 = net.addHost('h76', cls=Host, ip='192.168.8.76', defaultRoute=None)
h77 = net.addHost('h77', cls=Host, ip='192.168.8.77', defaultRoute=None)
h78 = net.addHost('h78', cls=Host, ip='192.168.8.78', defaultRoute=None)
h79 = net.addHost('h79', cls=Host, ip='192.168.8.79', defaultRoute=None)
h80 = net.addHost('h80', cls=Host, ip='192.168.8.80', defaultRoute=None)
h81 = net.addHost('h81', cls=Host, ip='192.168.8.81', defaultRoute=None)
h82 = net.addHost('h82', cls=Host, ip='192.168.8.82', defaultRoute=None)
h83 = net.addHost('h83', cls=Host, ip='192.168.8.83', defaultRoute=None)
h84 = net.addHost('h84', cls=Host, ip='192.168.8.84', defaultRoute=None)
h85 = net.addHost('h85', cls=Host, ip='192.168.8.85', defaultRoute=None)
h86 = net.addHost('h86', cls=Host, ip='192.168.8.86', defaultRoute=None)
h87 = net.addHost('h87', cls=Host, ip='192.168.8.87', defaultRoute=None)
h88 = net.addHost('h88', cls=Host, ip='192.168.8.88', defaultRoute=None)
h89 = net.addHost('h89', cls=Host, ip='192.168.8.89', defaultRoute=None)
h90 = net.addHost('h90', cls=Host, ip='192.168.8.90', defaultRoute=None)
h91 = net.addHost('h91', cls=Host, ip='192.168.8.91', defaultRoute=None)
h92 = net.addHost('h92', cls=Host, ip='192.168.8.92', defaultRoute=None)
h93 = net.addHost('h93', cls=Host, ip='192.168.8.93', defaultRoute=None)
h94 = net.addHost('h94', cls=Host, ip='192.168.8.94', defaultRoute=None)
h95 = net.addHost('h95', cls=Host, ip='192.168.8.95', defaultRoute=None)
h96 = net.addHost('h96', cls=Host, ip='192.168.8.96', defaultRoute=None)
h97 = net.addHost('h97', cls=Host, ip='192.168.8.97', defaultRoute=None)
h98 = net.addHost('h98', cls=Host, ip='192.168.8.98', defaultRoute=None)
h99 = net.addHost('h99', cls=Host, ip='192.168.8.99', defaultRoute=None)
h100 = net.addHost('h100', cls=Host, ip='192.168.8.100', defaultRoute=None)
h101 = net.addHost('h101', cls=Host, ip='192.168.8.101', defaultRoute=None)
h102 = net.addHost('h102', cls=Host, ip='192.168.8.102', defaultRoute=None)
h103 = net.addHost('h103', cls=Host, ip='192.168.8.103', defaultRoute=None)
h104 = net.addHost('h104', cls=Host, ip='192.168.8.104', defaultRoute=None)
h105 = net.addHost('h105', cls=Host, ip='192.168.8.105', defaultRoute=None)
h106 = net.addHost('h106', cls=Host, ip='192.168.8.106', defaultRoute=None)
h107 = net.addHost('h107', cls=Host, ip='192.168.8.107', defaultRoute=None)
h108 = net.addHost('h108', cls=Host, ip='192.168.8.108', defaultRoute=None)
h109 = net.addHost('h109', cls=Host, ip='192.168.8.109', defaultRoute=None)
h110 = net.addHost('h110', cls=Host, ip='192.168.8.110', defaultRoute=None)
h111 = net.addHost('h111', cls=Host, ip='192.168.8.111', defaultRoute=None)
h112 = net.addHost('h112', cls=Host, ip='192.168.8.112', defaultRoute=None)
h113 = net.addHost('h113', cls=Host, ip='192.168.8.113', defaultRoute=None)
h114 = net.addHost('h114', cls=Host, ip='192.168.8.114', defaultRoute=None)
h115 = net.addHost('h115', cls=Host, ip='192.168.8.115', defaultRoute=None)
h116 = net.addHost('h116', cls=Host, ip='192.168.8.116', defaultRoute=None)
h117 = net.addHost('h117', cls=Host, ip='192.168.8.117', defaultRoute=None)
```

```
h118 = net.addHost('h118', cls=Host, ip='192.168.8.118', defaultRoute=None)
h119 = net.addHost('h119', cls=Host, ip='192.168.8.119', defaultRoute=None)
h120 = net.addHost('h120', cls=Host, ip='192.168.8.120', defaultRoute=None)
h121 = net.addHost('h121', cls=Host, ip='192.168.8.121', defaultRoute=None)
h122 = net.addHost('h122', cls=Host, ip='192.168.8.122', defaultRoute=None)
h123 = net.addHost('h123', cls=Host, ip='192.168.8.123', defaultRoute=None)
h124 = net.addHost('h124', cls=Host, ip='192.168.8.124', defaultRoute=None)
h125 = net.addHost('h125', cls=Host, ip='192.168.8.125', defaultRoute=None)
h126 = net.addHost('h126', cls=Host, ip='192.168.8.126', defaultRoute=None)
h127 = net.addHost('h127', cls=Host, ip='192.168.8.127', defaultRoute=None)
h128 = net.addHost('h128', cls=Host, ip='192.168.8.128', defaultRoute=None)
h129 = net.addHost('h129', cls=Host, ip='192.168.8.129', defaultRoute=None)
h130 = net.addHost('h130', cls=Host, ip='192.168.8.130', defaultRoute=None)
h131 = net.addHost('h131', cls=Host, ip='192.168.8.131', defaultRoute=None)
h132 = net.addHost('h132', cls=Host, ip='192.168.8.132', defaultRoute=None)
h133 = net.addHost('h133', cls=Host, ip='192.168.8.133', defaultRoute=None)
h134 = net.addHost('h134', cls=Host, ip='192.168.8.134', defaultRoute=None)
h135 = net.addHost('h135', cls=Host, ip='192.168.8.135', defaultRoute=None)
h136 = net.addHost('h136', cls=Host, ip='192.168.8.136', defaultRoute=None)
h137 = net.addHost('h137', cls=Host, ip='192.168.8.137', defaultRoute=None)
h138 = net.addHost('h138', cls=Host, ip='192.168.8.138', defaultRoute=None)
h139 = net.addHost('h139', cls=Host, ip='192.168.8.139', defaultRoute=None)
h140 = net.addHost('h140', cls=Host, ip='192.168.8.140', defaultRoute=None)
h141 = net.addHost('h141', cls=Host, ip='192.168.8.141', defaultRoute=None)
h142 = net.addHost('h142', cls=Host, ip='192.168.8.142', defaultRoute=None)
h143 = net.addHost('h143', cls=Host, ip='192.168.8.143', defaultRoute=None)
h144 = net.addHost('h144', cls=Host, ip='192.168.8.144', defaultRoute=None)
h145 = net.addHost('h145', cls=Host, ip='192.168.8.145', defaultRoute=None)
h146 = net.addHost('h146', cls=Host, ip='192.168.8.146', defaultRoute=None)
h147 = net.addHost('h147', cls=Host, ip='192.168.8.147', defaultRoute=None)
h148 = net.addHost('h148', cls=Host, ip='192.168.8.148', defaultRoute=None)
h149 = net.addHost('h149', cls=Host, ip='192.168.8.149', defaultRoute=None)
h150 = net.addHost('h150', cls=Host, ip='192.168.8.150', defaultRoute=None)
h151 = net.addHost('h151', cls=Host, ip='192.168.8.151', defaultRoute=None)
h152 = net.addHost('h152', cls=Host, ip='192.168.8.152', defaultRoute=None)
h153 = net.addHost('h153', cls=Host, ip='192.168.8.153', defaultRoute=None)
h154 = net.addHost('h154', cls=Host, ip='192.168.8.154', defaultRoute=None)
h155 = net.addHost('h155', cls=Host, ip='192.168.8.155', defaultRoute=None)
h156 = net.addHost('h156', cls=Host, ip='192.168.8.156', defaultRoute=None)
h157 = net.addHost('h157', cls=Host, ip='192.168.8.157', defaultRoute=None)
h158 = net.addHost('h158', cls=Host, ip='192.168.8.158', defaultRoute=None)
h159 = net.addHost('h159', cls=Host, ip='192.168.8.159', defaultRoute=None)
h160 = net.addHost('h160', cls=Host, ip='192.168.8.160', defaultRoute=None)
h161 = net.addHost('h161', cls=Host, ip='192.168.8.161', defaultRoute=None)
h162 = net.addHost('h162', cls=Host, ip='192.168.8.162', defaultRoute=None)
h163 = net.addHost('h163', cls=Host, ip='192.168.8.163', defaultRoute=None)
h164 = net.addHost('h164', cls=Host, ip='192.168.8.164', defaultRoute=None)
h165 = net.addHost('h165', cls=Host, ip='192.168.8.165', defaultRoute=None)
h166 = net.addHost('h166', cls=Host, ip='192.168.8.166', defaultRoute=None)
h167 = net.addHost('h167', cls=Host, ip='192.168.8.167', defaultRoute=None)
```

```
h168 = net.addHost('h168', cls=Host, ip='192.168.8.168', defaultRoute=None)
h169 = net.addHost('h169', cls=Host, ip='192.168.8.169', defaultRoute=None)
h170 = net.addHost('h170', cls=Host, ip='192.168.8.170', defaultRoute=None)
h171 = net.addHost('h171', cls=Host, ip='192.168.8.171', defaultRoute=None)
h172 = net.addHost('h172', cls=Host, ip='192.168.8.172', defaultRoute=None)
h173 = net.addHost('h173', cls=Host, ip='192.168.8.173', defaultRoute=None)
h174 = net.addHost('h174', cls=Host, ip='192.168.8.174', defaultRoute=None)
h175 = net.addHost('h175', cls=Host, ip='192.168.8.175', defaultRoute=None)
h176 = net.addHost('h176', cls=Host, ip='192.168.8.176', defaultRoute=None)
h177 = net.addHost('h177', cls=Host, ip='192.168.8.177', defaultRoute=None)
h178 = net.addHost('h178', cls=Host, ip='192.168.8.178', defaultRoute=None)
h179 = net.addHost('h179', cls=Host, ip='192.168.8.179', defaultRoute=None)
h180 = net.addHost('h180', cls=Host, ip='192.168.8.180', defaultRoute=None)
h181 = net.addHost('h181', cls=Host, ip='192.168.8.181', defaultRoute=None)
h182 = net.addHost('h182', cls=Host, ip='192.168.8.182', defaultRoute=None)
h183 = net.addHost('h183', cls=Host, ip='192.168.8.183', defaultRoute=None)
h184 = net.addHost('h184', cls=Host, ip='192.168.8.184', defaultRoute=None)
h185 = net.addHost('h185', cls=Host, ip='192.168.8.185', defaultRoute=None)
h186 = net.addHost('h186', cls=Host, ip='192.168.8.186', defaultRoute=None)
h187 = net.addHost('h187', cls=Host, ip='192.168.8.187', defaultRoute=None)
h188 = net.addHost('h188', cls=Host, ip='192.168.8.188', defaultRoute=None)
h189 = net.addHost('h189', cls=Host, ip='192.168.8.189', defaultRoute=None)
h190 = net.addHost('h190', cls=Host, ip='192.168.8.190', defaultRoute=None)
h191 = net.addHost('h191', cls=Host, ip='192.168.8.191', defaultRoute=None)
h192 = net.addHost('h192', cls=Host, ip='192.168.8.192', defaultRoute=None)
h193 = net.addHost('h193', cls=Host, ip='192.168.8.193', defaultRoute=None)
h194 = net.addHost('h194', cls=Host, ip='192.168.8.194', defaultRoute=None)
h195 = net.addHost('h195', cls=Host, ip='192.168.8.195', defaultRoute=None)
h196 = net.addHost('h196', cls=Host, ip='192.168.8.196', defaultRoute=None)
h197 = net.addHost('h197', cls=Host, ip='192.168.8.197', defaultRoute=None)
h198 = net.addHost('h198', cls=Host, ip='192.168.8.198', defaultRoute=None)
h199 = net.addHost('h199', cls=Host, ip='192.168.8.199', defaultRoute=None)
h200 = net.addHost('h200', cls=Host, ip='192.168.8.200', defaultRoute=None)
h201 = net.addHost('h201', cls=Host, ip='192.168.8.201', defaultRoute=None)
h202 = net.addHost('h202', cls=Host, ip='192.168.8.202', defaultRoute=None)
h203 = net.addHost('h203', cls=Host, ip='192.168.8.203', defaultRoute=None)
h204 = net.addHost('h204', cls=Host, ip='192.168.8.204', defaultRoute=None)
h205 = net.addHost('h205', cls=Host, ip='192.168.8.205', defaultRoute=None)
h206 = net.addHost('h206', cls=Host, ip='192.168.8.206', defaultRoute=None)
h207 = net.addHost('h207', cls=Host, ip='192.168.8.207', defaultRoute=None)
h208 = net.addHost('h208', cls=Host, ip='192.168.8.208', defaultRoute=None)
h209 = net.addHost('h209', cls=Host, ip='192.168.8.209', defaultRoute=None)
h210 = net.addHost('h210', cls=Host, ip='192.168.8.210', defaultRoute=None)
h211 = net.addHost('h211', cls=Host, ip='192.168.8.211', defaultRoute=None)
h212 = net.addHost('h212', cls=Host, ip='192.168.8.212', defaultRoute=None)
h213 = net.addHost('h213', cls=Host, ip='192.168.8.213', defaultRoute=None)
h214 = net.addHost('h214', cls=Host, ip='192.168.8.214', defaultRoute=None)
h215 = net.addHost('h215', cls=Host, ip='192.168.8.215', defaultRoute=None)
h216 = net.addHost('h216', cls=Host, ip='192.168.8.216', defaultRoute=None)
h217 = net.addHost('h217', cls=Host, ip='192.168.8.217', defaultRoute=None)
```

```
    h218 = net.addHost('h218', cls=Host, ip='192.168.8.218', defaultRoute=None)
    h219 = net.addHost('h219', cls=Host, ip='192.168.8.219', defaultRoute=None)
    h220 = net.addHost('h220', cls=Host, ip='192.168.8.220', defaultRoute=None)
    h221 = net.addHost('h221', cls=Host, ip='192.168.8.221', defaultRoute=None)
    h222 = net.addHost('h222', cls=Host, ip='192.168.8.222', defaultRoute=None)
    h223 = net.addHost('h223', cls=Host, ip='192.168.8.223', defaultRoute=None)
    h224 = net.addHost('h224', cls=Host, ip='192.168.8.224', defaultRoute=None)
    h225 = net.addHost('h225', cls=Host, ip='192.168.8.225', defaultRoute=None)
    h226 = net.addHost('h226', cls=Host, ip='192.168.8.226', defaultRoute=None)
    h227 = net.addHost('h227', cls=Host, ip='192.168.8.227', defaultRoute=None)
    h228 = net.addHost('h228', cls=Host, ip='192.168.8.228', defaultRoute=None)
    h229 = net.addHost('h229', cls=Host, ip='192.168.8.229', defaultRoute=None)
    h230 = net.addHost('h230', cls=Host, ip='192.168.8.230', defaultRoute=None)
    h231 = net.addHost('h231', cls=Host, ip='192.168.8.231', defaultRoute=None)
    h232 = net.addHost('h232', cls=Host, ip='192.168.8.232', defaultRoute=None)
    h233 = net.addHost('h233', cls=Host, ip='192.168.8.233', defaultRoute=None)
    h234 = net.addHost('h234', cls=Host, ip='192.168.8.234', defaultRoute=None)
    h235 = net.addHost('h235', cls=Host, ip='192.168.8.235', defaultRoute=None)
    h236 = net.addHost('h236', cls=Host, ip='192.168.8.236', defaultRoute=None)
    h237 = net.addHost('h237', cls=Host, ip='192.168.8.237', defaultRoute=None)
    h238 = net.addHost('h238', cls=Host, ip='192.168.8.238', defaultRoute=None)
    h239 = net.addHost('h239', cls=Host, ip='192.168.8.239', defaultRoute=None)
#   h240 = net.addHost('h240', cls=Host, ip='192.168.8.240', defaultRoute=None)
#   h241 = net.addHost('h241', cls=Host, ip='192.168.8.241', defaultRoute=None)
#   h242 = net.addHost('h242', cls=Host, ip='192.168.8.242', defaultRoute=None)
#   h243 = net.addHost('h243', cls=Host, ip='192.168.8.243', defaultRoute=None)
#   h244 = net.addHost('h244', cls=Host, ip='192.168.8.244', defaultRoute=None)
#   h245 = net.addHost('h245', cls=Host, ip='192.168.8.245', defaultRoute=None)
#   h246 = net.addHost('h246', cls=Host, ip='192.168.8.246', defaultRoute=None)
#   h247 = net.addHost('h247', cls=Host, ip='192.168.8.247', defaultRoute=None)
#   h248 = net.addHost('h248', cls=Host, ip='192.168.8.248', defaultRoute=None)
#   h249 = net.addHost('h249', cls=Host, ip='192.168.8.249', defaultRoute=None)
#   h250 = net.addHost('h250', cls=Host, ip='192.168.8.250', defaultRoute=None)
#   h251 = net.addHost('h251', cls=Host, ip='192.168.8.251', defaultRoute=None)
    h252 = net.addHost('h252', cls=Host, ip='192.168.8.252', defaultRoute=None)
    server1 = net.addHost('server1', cls=Host, ip='192.168.8.253', defaultRoute=None)
    info( '*** Add links\n')
#   net.addLink(h1, s2)
#   net.addLink(h2, s2)
#   net.addLink(h3, s2)
#   net.addLink(h4, s2)
#   net.addLink(h5, s2)
#   net.addLink(h6, s2)
#   net.addLink(h7, s2)
#   net.addLink(h8, s2)
#   net.addLink(h9, s2)
#   net.addLink(h10, s2)
#   net.addLink(h11, s2)
#   net.addLink(h12, s2)
    net.addLink(h13, s2)
```

```
net.addLink(h14, s2)
net.addLink(h15, s2)
net.addLink(h16, s2)
net.addLink(h17, s2)
net.addLink(h18, s2)
net.addLink(h19, s2)
net.addLink(h20, s2)
net.addLink(h21, s2)
net.addLink(h22, s2)
net.addLink(h23, s2)
net.addLink(h24, s2)
net.addLink(h25, s2)
net.addLink(h26, s2)
net.addLink(h27, s2)
net.addLink(h28, s2)
net.addLink(h29, s2)
net.addLink(h30, s2)
net.addLink(h31, s2)
net.addLink(h32, s2)
net.addLink(h33, s3)
net.addLink(h34, s3)
net.addLink(h35, s3)
net.addLink(h36, s3)
net.addLink(h37, s3)
net.addLink(h38, s3)
net.addLink(h39, s3)
net.addLink(h40, s3)
net.addLink(h41, s3)
net.addLink(h42, s3)
net.addLink(h43, s3)
net.addLink(h44, s3)
net.addLink(h45, s3)
net.addLink(h46, s3)
net.addLink(h47, s3)
net.addLink(h48, s3)
net.addLink(h49, s3)
net.addLink(h50, s3)
net.addLink(h51, s3)
net.addLink(h52, s3)
net.addLink(h53, s3)
net.addLink(h54, s3)
net.addLink(h55, s3)
net.addLink(h56, s3)
net.addLink(h57, s3)
net.addLink(h58, s3)
net.addLink(h59, s3)
net.addLink(h60, s3)
net.addLink(h61, s3)
net.addLink(h62, s3)
net.addLink(h63, s3)
```

```
net.addLink(h64, s3)
net.addLink(h65, s4)
net.addLink(h66, s4)
net.addLink(h67, s4)
net.addLink(h68, s4)
net.addLink(h69, s4)
net.addLink(h70, s4)
net.addLink(h71, s4)
net.addLink(h72, s4)
net.addLink(h73, s4)
net.addLink(h74, s4)
net.addLink(h75, s4)
net.addLink(h76, s4)
net.addLink(h77, s4)
net.addLink(h78, s4)
net.addLink(h79, s4)
net.addLink(h80, s4)
net.addLink(h81, s4)
net.addLink(h82, s4)
net.addLink(h83, s4)
net.addLink(h84, s4)
net.addLink(h85, s4)
net.addLink(h86, s4)
net.addLink(h87, s4)
net.addLink(h88, s4)
net.addLink(h89, s4)
net.addLink(h90, s4)
net.addLink(h91, s4)
net.addLink(h92, s4)
net.addLink(h93, s4)
net.addLink(h94, s4)
net.addLink(h95, s4)
net.addLink(h96, s4)
net.addLink(h97, s5)
net.addLink(h98, s5)
net.addLink(h99, s5)
net.addLink(h100, s5)
net.addLink(h101, s5)
net.addLink(h102, s5)
net.addLink(h103, s5)
net.addLink(h104, s5)
net.addLink(h105, s5)
net.addLink(h106, s5)
net.addLink(h107, s5)
net.addLink(h108, s5)
net.addLink(h109, s5)
net.addLink(h110, s5)
net.addLink(h111, s5)
net.addLink(h112, s5)
net.addLink(h113, s5)
```

```
net.addLink(h114, s5)
net.addLink(h115, s5)
net.addLink(h116, s5)
net.addLink(h117, s5)
net.addLink(h118, s5)
net.addLink(h119, s5)
net.addLink(h120, s5)
net.addLink(h121, s5)
net.addLink(h122, s5)
net.addLink(h123, s5)
net.addLink(h124, s5)
net.addLink(h125, s5)
net.addLink(h126, s5)
net.addLink(h127, s5)
net.addLink(h128, s5)
net.addLink(h129, s6)
net.addLink(h130, s6)
net.addLink(h131, s6)
net.addLink(h132, s6)
net.addLink(h133, s6)
net.addLink(h134, s6)
net.addLink(h135, s6)
net.addLink(h136, s6)
net.addLink(h137, s6)
net.addLink(h138, s6)
net.addLink(h139, s6)
net.addLink(h140, s6)
net.addLink(h141, s6)
net.addLink(h142, s6)
net.addLink(h143, s6)
net.addLink(h144, s6)
net.addLink(h145, s6)
net.addLink(h146, s6)
net.addLink(h147, s6)
net.addLink(h148, s6)
net.addLink(h149, s6)
net.addLink(h150, s6)
net.addLink(h151, s6)
net.addLink(h152, s6)
net.addLink(h153, s6)
net.addLink(h154, s6)
net.addLink(h155, s6)
net.addLink(h156, s6)
net.addLink(h157, s6)
net.addLink(h158, s6)
net.addLink(h159, s6)
net.addLink(h160, s6)
net.addLink(h161, s7)
net.addLink(h162, s7)
net.addLink(h163, s7)
```

```
net.addLink(h164, s7)
net.addLink(h165, s7)
net.addLink(h166, s7)
net.addLink(h167, s7)
net.addLink(h168, s7)
net.addLink(h169, s7)
net.addLink(h170, s7)
net.addLink(h171, s7)
net.addLink(h172, s7)
net.addLink(h173, s7)
net.addLink(h174, s7)
net.addLink(h175, s7)
net.addLink(h176, s7)
net.addLink(h177, s7)
net.addLink(h178, s7)
net.addLink(h179, s7)
net.addLink(h180, s7)
net.addLink(h181, s7)
net.addLink(h182, s7)
net.addLink(h183, s7)
net.addLink(h184, s7)
net.addLink(h185, s7)
net.addLink(h186, s7)
net.addLink(h187, s7)
net.addLink(h188, s7)
net.addLink(h189, s7)
net.addLink(h190, s7)
net.addLink(h191, s7)
net.addLink(h192, s7)
net.addLink(h193, s8)
net.addLink(h194, s8)
net.addLink(h195, s8)
net.addLink(h196, s8)
net.addLink(h197, s8)
net.addLink(h198, s8)
net.addLink(h199, s8)
net.addLink(h200, s8)
net.addLink(h201, s8)
net.addLink(h202, s8)
net.addLink(h203, s8)
net.addLink(h204, s8)
net.addLink(h205, s8)
net.addLink(h206, s8)
net.addLink(h207, s8)
net.addLink(h208, s8)
net.addLink(h209, s8)
net.addLink(h210, s8)
net.addLink(h211, s8)
net.addLink(h212, s8)
net.addLink(h213, s8)
```

```
    net.addLink(h214, s8)
    net.addLink(h215, s8)
    net.addLink(h216, s8)
    net.addLink(h217, s8)
    net.addLink(h218, s8)
    net.addLink(h219, s8)
    net.addLink(h220, s8)
    net.addLink(h221, s8)
    net.addLink(h222, s8)
    net.addLink(h223, s8)
    net.addLink(h224, s8)
    net.addLink(h225, s9)
    net.addLink(h226, s9)
    net.addLink(h227, s9)
    net.addLink(h228, s9)
    net.addLink(h229, s9)
    net.addLink(h230, s9)
    net.addLink(h231, s9)
    net.addLink(h232, s9)
    net.addLink(h233, s9)
    net.addLink(h234, s9)
    net.addLink(h235, s9)
    net.addLink(h236, s9)
    net.addLink(h237, s9)
    net.addLink(h238, s9)
    net.addLink(h239, s9)
#   net.addLink(h240, s9)
#   net.addLink(h241, s9)
#   net.addLink(h242, s9)
#   net.addLink(h243, s9)
#   net.addLink(h244, s9)
#   net.addLink(h245, s9)
#   net.addLink(h246, s9)
#   net.addLink(h247, s9)
#   net.addLink(h248, s9)
#   net.addLink(h249, s9)
#   net.addLink(h250, s9)
#   net.addLink(h251, s9)
    net.addLink(h252, s9)
    net.addLink(s1, server1)
    net.addLink(s1, s9)
    net.addLink(s9, s8)
    net.addLink(s8, s7)
    net.addLink(s7, s6)
    net.addLink(s6, s5)
    net.addLink(s5, s4)
    net.addLink(s4, s3)
    net.addLink(s3, s2)
    info( '*** Starting network\n')
    net.build()
```

```
    info( '*** Starting controllers\n')
    for controller in net.controllers:
       controller.start()
    info( '*** Starting switches\n')
    net.get('s1').start([c0])
    net.get('s2').start([c0])
    net.get('s3').start([c0])
    net.get('s4').start([c0])
    net.get('s5').start([c0])
    net.get('s6').start([c0])
    net.get('s7').start([c0])
    net.get('s8').start([c0])
    net.get('s9').start([c0])
    info( '*** Post configure switches and hosts\n')
    CLI(net)
    net.stop()
if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

Appendix B: Background-Traffic and Bandwidth-Limit Dataset Exploration using Jupyter in Anaconda3

In [2]:

```
## Data flow discription
# 0   Treatment      = TR
# 1   switch         = SW
# 2   Time(s)        = T
# 3   packetsNumber  = PN
# 4   bytesNumber    = BN
# 5   idleAge        = IA
# 6   Protocol       = PR
# 7   inPort         = PI
# 8   Raw IP flow, ARP_spa, Nw_src = RIFS
# 9   arp & nw SOURCE 1 for ARP_spa, 0 for Nw_src = ANSS
# 10  FLOW, 1 for ARP_spa, 0 for Nw_src = FS
# 11  Raw IP flow, ARP_spa, Nw_dst = RIFD
# 12  arp & network SOURCE 1 for ARP_spa, 0 for Nw_dst =ANSD
# 13  FLOW, 1 for ARP_spa, 0 for Nw_dst = FD
```

In [3]:

```
#import the neede libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import VotingClassifier
from sklearn.feature_selection import VarianceThreshold

from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import f1_score

import seaborn as sns
import matplotlib.pyplot as plt

# Comment this if the data visualisations doesn't work on your side
%matplotlib inline

plt.style.use('bmh')
```

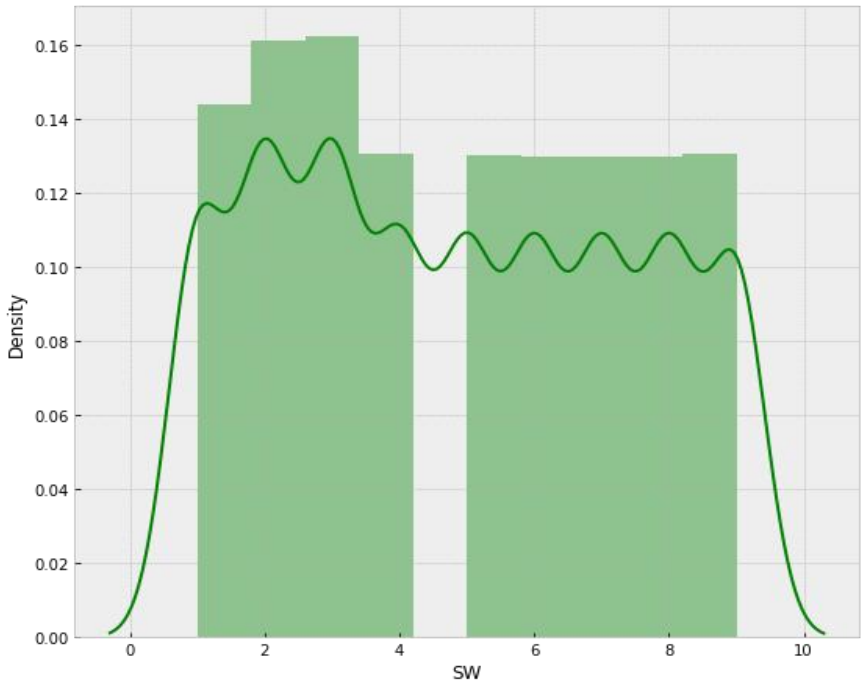
In [4]:

```
#load the dataset
df=pd.read_csv('data-LANS-SDN-INTERNET/T1U2.csv')
```

```
#len(df)
df.shape
```
Out[4]:
```
(7772, 15)
```
In [5]:
```
i=df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7772 entries, 0 to 7771
Data columns (total 15 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   T1      7772 non-null   int64  
 1   U2      7772 non-null   int64  
 2   SW      7772 non-null   int64  
 3   T       7772 non-null   float64
 4   PN      7772 non-null   int64  
 5   BN      7772 non-null   int64  
 6   IA      7772 non-null   int64  
 7   PR      7772 non-null   object 
 8   PI      7772 non-null   int64  
 9   RIFS    7772 non-null   object 
 10  ANSS    7772 non-null   object 
 11  FS      7772 non-null   int64  
 12  RIFD    7772 non-null   object 
 13  ANSD    7772 non-null   object 
 14  FD      7772 non-null   int64  
dtypes: float64(1), int64(9), object(5)
memory usage: 910.9+ KB
```
In [6]:
```
q=df
#q.to_csv('T1U1-dataflow-info.csv')
```

7772 rows × 15 columns

In [7]:
```
print(df['SW'].describe())
plt.figure(figsize=(9, 8))
sns.distplot(df['SW'], color='g', bins=10, hist_kws={'alpha': 0.4});
count    7772.000000
mean        4.829645
std         2.594314
min         1.000000
25%         3.000000
50%         5.000000
75%         7.000000
max         9.000000
Name: SW, dtype: float64
/home/olonlonse/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [8]:

```
i=df.describe()
# rounding the number upto 2 decimal places and save to s csv
round(i, 2).to_csv('results-for-T1U1/T1U2-mean-effect.csv')
print(round(i, 2))
```

```
           T1      U2       SW        T          PN            BN       IA  \
count  7772.0  7772.0  7772.00  7772.00    7772.00  7.772000e+03  7772.00
mean     64.0   256.0     4.83     7.90    1643.13  2.467972e+06     3.18
std       0.0     0.0     2.59     6.63   10390.45  1.571807e+07     3.03
min      64.0   256.0     1.00     0.00       1.00  4.200000e+01     0.00
25%      64.0   256.0     3.00     3.44       1.00  7.400000e+01     0.00
50%      64.0   256.0     5.00     6.11       2.00  1.710000e+02     3.00
75%      64.0   256.0     7.00     9.52      14.00  1.800000e+03     6.00
max      64.0   256.0     9.00    30.02  101899.00  1.540713e+08    10.00

            PI       FS       FD
count  7772.00  7772.00  7772.00
mean     26.23     0.12     0.12
std       8.36     0.32     0.32
min       2.00     0.00     0.00
25%      29.00     0.00     0.00
50%      29.00     0.00     0.00
75%      30.00     0.00     0.00
max      30.00     1.00     1.00
```

In [9]:

```
rows_without_missing_data = df.dropna()
rows_without_missing_data.shape
```

Out[9]:

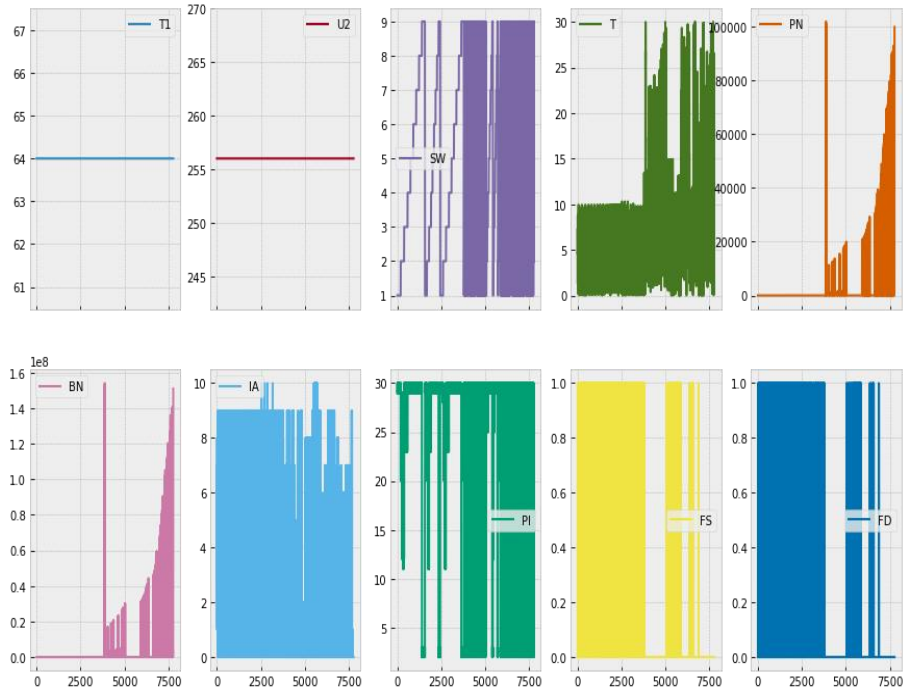
```
(7772, 15)
```

In [10]:

```
df.plot(kind='line', subplots=True, layout=(2,5), figsize=(15,10), title='')
```

Out[10]:

```
array([[<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
        <AxesSubplot:>],
       [<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```

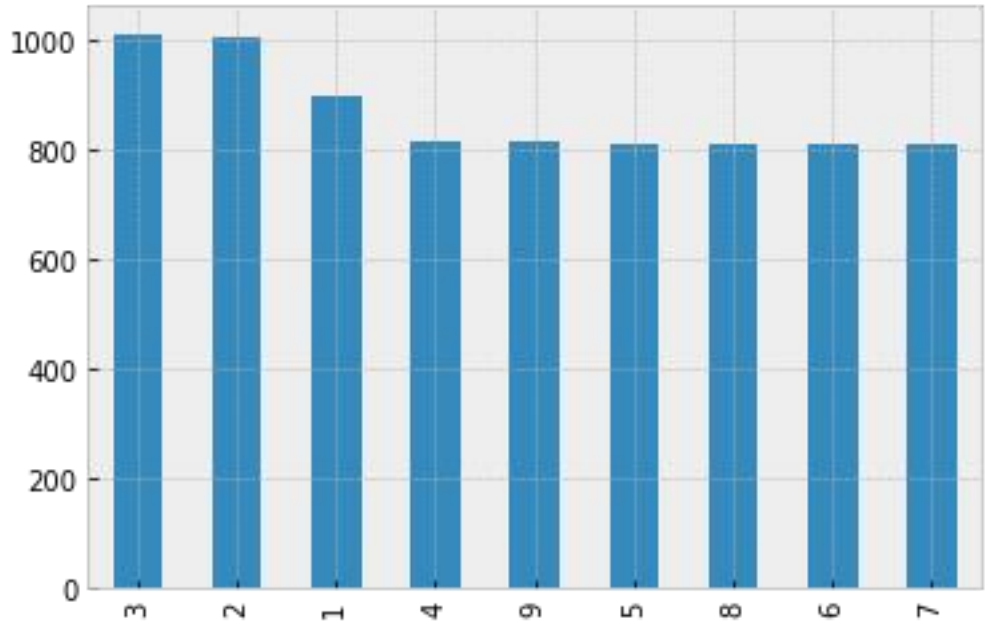


In [11]:

```
%matplotlib inline
df["SW"].value_counts().plot(kind="bar")
```

Out[11]:

```
<AxesSubplot:>
```



In [12]:

```
list(set(df.dtypes.tolist()))
```

Out[12]:

```
[dtype('O'), dtype('int64'), dtype('float64')]
```

In [13]:

```
df_num = df.select_dtypes(include = ['float64', 'int64'])
df_num.head()
df_num
```

Out[13]:

	T1	U2	SW	T	PN	BN	IA	PI	FS	FD
0	64	256	1	4.684	1	91	4	29	0	0
1	64	256	1	4.549	1	141	4	30	0	0
2	64	256	1	7.324	1	42	7	30	1	1
3	64	256	1	7.317	1	42	7	29	1	1
4	64	256	1	5.174	1	42	5	29	1	1
...	...	...	...	...	...	...	...	...	...	...
7767	64	256	8	26.229	99526	150483312	0	29	0	0
7768	64	256	5	11.203	996	2741640	0	29	0	0
7769	64	256	9	26.268	99640	150655680	0	2	0	0
7770	64	256	3	26.435	99741	150808392	0	29	0	0
7771	64	256	2	26.469	99844	150964128	0	12	0	0

7772 rows × 10 columns

In [14]:

```
df_num.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8); # ; avoid
having the matplotlib verbose informations
```

In [15]:

```
df_num_corr = df_num.corr()
#round(inputNumber, 2)
round(df_num_corr, 2).to_csv('results-for-T1U1/T1U2-corr.csv')
round(df_num_corr, 2)
```

Out[15]:

	T1	U2	SW	T	PN	BN	IA	PI	FS	FD
T1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
U2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SW	NaN	NaN	1.00	-0.02	0.04	0.04	0.02	-0.49	-0.07	-0.07
T	NaN	NaN	-0.02	1.00	0.30	0.30	-0.07	-0.02	-0.15	-0.15
PN	NaN	NaN	0.04	0.30	1.00	1.00	-0.17	-0.05	-0.06	-0.06
BN	NaN	NaN	0.04	0.30	1.00	1.00	-0.16	-0.05	-0.06	-0.06
IA	NaN	NaN	0.02	-0.07	-0.17	-0.16	1.00	0.01	0.19	0.19
PI	NaN	NaN	-0.49	-0.02	-0.05	-0.05	0.01	1.00	-0.00	-0.00
FS	NaN	NaN	-0.07	-0.15	-0.06	-0.06	0.19	-0.00	1.00	1.00
FD	NaN	NaN	-0.07	-0.15	-0.06	-0.06	0.19	-0.00	1.00	1.00

In [16]:

```
df_num_corr = df_num.corr()['PN'][:-1] # -1 because the latest row is SalePrice
golden_features_list = df_num_corr[abs(df_num_corr) >
0.5].sort_values(ascending=False)
print("The {} strongly correlated values with
idleAge:\n{}".format(len(golden_features_list), golden_features_list))
The 2 strongly correlated values with idleAge:
PN    1.000000
BN    0.999646
Name: PN, dtype: float64
```

In [29]:

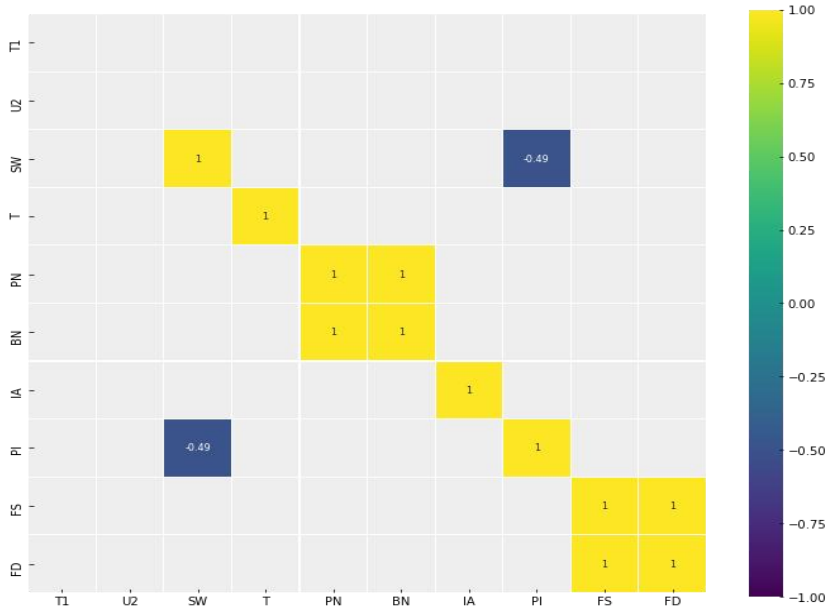
```
#for i in range(0, len(df_num.columns), 2):
#    sns.pairplot(data=df_num,
#                 x_vars=df_num.columns[i:i+1],
```

```
#                 y_vars=['BN'])
```

In [30]:

```
corr = df_num.corr() # We already examined  correlations
plt.figure(figsize=(12, 10))

sns.heatmap(corr[(corr >= 0.5) | (corr <= -0.4)],
            cmap='viridis', vmax=1.0, vmin=-1.0, linewidths=0.1,
            annot=True, annot_kws={"size": 8}, square=True);
```



In [ ]:

Appendix C: 2-Stack Model for the dataset on Background-Traffic and Bandwidth-Limit

Experiment

In [1]:
```
#T1=64 bytes
#T2=58956 bytes
#T3=65507 bytes
#
#U1=1M
#U2=256M
#U3=512M
```
In [2]:
```
#import the neede libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import VotingClassifier
from sklearn.feature_selection import VarianceThreshold

from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import f1_score

import seaborn as sns
import matplotlib.pyplot as plt

# Comment this if the data visualisations doesn't work on your side
%matplotlib inline

plt.style.use('bmh')
```
In [3]:
```
#load the dataset
#df = pd.read_csv('data-LANS-SDN-INTERNET/T1U1.csv')

#load the dataset
df=pd.read_csv('data-LANS-SDN-INTERNET/T1U1.csv')

#len(df)
df.shape
```
Out[3]:
```
(10949, 15)
```
In [4]:
```
df
```
Out[4]:

	T1	U1	SW	T	PN	BN	IA	PR	PI	RIFS	ANSS	FS	
0	64	1.0	2	0.008	1	42	0	arp	29	arp_spa=192.168.8.81	192.168.8.81	1	arp_ 8.8.1
1	64	1.0	2	0.013	1	42	0	arp	12	arp_spa=192.168.8.42	192.168.8.42	1	arp_ 8.8.2
2	64	1.0	3	0.015	1	42	0	arp	24	arp_spa=192.168.8.81	192.168.8.81	1	arp_ 8.8.1
3	64	1.0	3	0.015	1	42	0	arp	29	arp_spa=192.168.8.42	192.168.8.42	1	arp_ 8.8.2
4	64	1.0	4	0.017	1	42	0	arp	29	arp_spa=192.168.8.42	192.168.8.42	1	arp_ 8.8.2

	T1	U1	SW	T	PN	BN	IA	PR	PI	RIFS	ANSS	FS	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
10944	64	1.0	5	29.276	91986	139082832	0	udp	29	nw_src=192.168.8.42	192.168.8.42	0	nw_ .8.22
10945	64	1.0	7	12.760	92	90699	0	tcp	29	nw_src=64.233.184.109	64.233.184.109	0	nw_ .8.22
10946	64	1.0	6	14.076	922	2444260	0	tcp	29	nw_src=192.168.8.41	192.168.8.41	0	nw_ .8.22
10947	64	1.0	9	12.762	94	92925	0	tcp	2	nw_src=64.233.184.109	64.233.184.109	0	nw_ .8.22
10948	64	1.0	8	29.679	94805	143345160	0	udp	29	nw_src=192.168.8.42	192.168.8.42	0	nw_ .8.22

10949 rows × 15 columns

In [5]:

```
list(set(df.dtypes.tolist()))
```

Out[5]:

```
[dtype('float64'), dtype('O'), dtype('int64')]
```

In [6]:

```
df_num = df.select_dtypes(include = ['float64', 'int64'])
df_num.head()
df_num
```

Out[6]:

	T1	U1	SW	T	PN	BN	IA	PI	FS	FD
0	64	1.0	2	0.008	1	42	0	29	1	1
1	64	1.0	2	0.013	1	42	0	12	1	1
2	64	1.0	3	0.015	1	42	0	24	1	1
3	64	1.0	3	0.015	1	42	0	29	1	1
4	64	1.0	4	0.017	1	42	0	29	1	1
...	...	...	...	...	...	...	...	...	...	...
10944	64	1.0	5	29.276	91986	139082832	0	29	0	0
10945	64	1.0	7	12.760	92	90699	0	29	0	0
10946	64	1.0	6	14.076	922	2444260	0	29	0	0
10947	64	1.0	9	12.762	94	92925	0	2	0	0
10948	64	1.0	8	29.679	94805	143345160	0	29	0	0

10949 rows × 10 columns

In [7]:

```
#using Protocol as target variable
X=df.drop(columns=['PR','ANSD', 'RIFD','ANSS','RIFS'],  axis=1)
y=df['PR']
```

In [8]:

```
X
```

Out[8]:

	T1	U1	SW	T	PN	BN	IA	PI	FS	FD
0	64	1.0	2	0.008	1	42	0	29	1	1
1	64	1.0	2	0.013	1	42	0	12	1	1
2	64	1.0	3	0.015	1	42	0	24	1	1
3	64	1.0	3	0.015	1	42	0	29	1	1
4	64	1.0	4	0.017	1	42	0	29	1	1
...	...	...	...	...	...	...	...	...	...	...
10944	64	1.0	5	29.276	91986	139082832	0	29	0	0
10945	64	1.0	7	12.760	92	90699	0	29	0	0

	T1	U1	SW	T	PN	BN	IA	PI	FS	FD
10946	64	1.0	6	14.076	922	2444260	0	29	0	0
10947	64	1.0	9	12.762	94	92925	0	2	0	0
10948	64	1.0	8	29.679	94805	143345160	0	29	0	0

10949 rows × 10 columns

In [9]:

```
selection = VarianceThreshold(threshold=(0.1))
X = selection.fit_transform(X)
X.shape
```

Out[9]:

```
(10949, 8)
```

In [10]:

```
X_train, X_test, y_train, y_test = train_test_split(
X, y, stratify=y, test_size=0.35, random_state=42)

X_train.shape, X_test.shape
```

Out[10]:

```
((7116, 8), (3833, 8))
```

In [11]:

```
y_train.value_counts()
```

Out[11]:

```
tcp     2361
icmp    1647
arp     1646
udp     1462
Name: PR, dtype: int64
```

In [12]:

```
y_test.value_counts()
```

Out[12]:

```
tcp     1272
icmp     888
arp      886
udp      787
Name: PR, dtype: int64
```

In [13]:

```
#Build Classification models

from sklearn.metrics import accuracy_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import f1_score
```

In [14]:

```
#K nearest neighbors

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(3) # Define classifier
knn.fit(X_train, y_train) # Train model

# Make predictions
y_train_pred = knn.predict(X_train)
y_test_pred = knn.predict(X_test)

# Training set performance
knn_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
knn_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
knn_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate F1-
score


# Test set performance
knn_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
knn_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
```

```
knn_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % knn_train_accuracy)
print('- MCC: %s' % knn_train_mcc)
print('- F1 score: %s' % knn_train_f1)
print('---------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % knn_test_accuracy)
print('- MCC: %s' % knn_test_mcc)
print('- F1 score: %s' % knn_test_f1)
```

```
Model performance for Training set
- Accuracy: 0.9799044406970208
- MCC: 0.9728832016332284
- F1 score: 0.9799160124647879
---------------------------------
Model performance for Test set
- Accuracy: 0.9663448995564832
- MCC: 0.9546482883106525
- F1 score: 0.9663757244554873
```

In [15]:

```
#Support vector machine (Radial basis function kernel)

from sklearn.svm import SVC

svm_rbf = SVC(gamma=2, C=1)
svm_rbf.fit(X_train, y_train)

# Make predictions
y_train_pred = svm_rbf.predict(X_train)
y_test_pred = svm_rbf.predict(X_test)

# Training set performance
svm_rbf_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
svm_rbf_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
svm_rbf_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate
F1-score

# Test set performance
svm_rbf_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
svm_rbf_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
svm_rbf_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-
score

print('Model performance for Training set')
print('- Accuracy: %s' % svm_rbf_train_accuracy)
print('- MCC: %s' % svm_rbf_train_mcc)
print('- F1 score: %s' % svm_rbf_train_f1)
print('---------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % svm_rbf_test_accuracy)
print('- MCC: %s' % svm_rbf_test_mcc)
print('- F1 score: %s' % svm_rbf_test_f1)
```

```
Model performance for Training set
- Accuracy: 0.9998594716132658
- MCC: 0.999810290548095
- F1 score: 0.999859480751884
---------------------------------
Model performance for Test set
- Accuracy: 0.7685885729193843
- MCC: 0.7205294173458786
- F1 score: 0.7482163114218551
```

In [16]:

```
#Decision tree

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(max_depth=5) # Define classifier
```

```
dt.fit(X_train, y_train) # Train model

# Make predictions
y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)

# Training set performance
dt_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
dt_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
dt_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate F1-
score

# Test set performance
dt_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
dt_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
dt_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % dt_train_accuracy)
print('- MCC: %s' % dt_train_mcc)
print('- F1 score: %s' % dt_train_f1)
print('---------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % dt_test_accuracy)
print('- MCC: %s' % dt_test_mcc)
print('- F1 score: %s' % dt_test_f1)
Model performance for Training set
- Accuracy: 0.898397976391231
- MCC: 0.8665683088153157
- F1 score: 0.8973465038587761
---------------------------------
Model performance for Test set
- Accuracy: 0.8912079311244456
- MCC: 0.8567556597007966
- F1 score: 0.8899898031351843
```

In [17]:

```
#Random forest

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=10) # Define classifier
rf.fit(X_train, y_train) # Train model

# Make predictions
y_train_pred = rf.predict(X_train)
y_test_pred = rf.predict(X_test)

# Training set performance
rf_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
rf_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
rf_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate F1-
score

# Test set performance
rf_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
rf_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
rf_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % rf_train_accuracy)
print('- MCC: %s' % rf_train_mcc)
print('- F1 score: %s' % rf_train_f1)
print('---------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % rf_test_accuracy)
print('- MCC: %s' % rf_test_mcc)
print('- F1 score: %s' % rf_test_f1)
Model performance for Training set
```

```
- Accuracy: 0.9998594716132658
- MCC: 0.9998102814541462
- F1 score: 0.9998594624519087
----------------------------------
Model performance for Test set
- Accuracy: 0.9861727106704931
- MCC: 0.981328007408551
- F1 score: 0.9861697788788655
```

In [18]:

```
#Neural network

from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(alpha=1, max_iter=1000)
mlp.fit(X_train, y_train)

# Make predictions
y_train_pred = mlp.predict(X_train)
y_test_pred = mlp.predict(X_test)

# Training set performance
mlp_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
mlp_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
mlp_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate F1-
score

# Test set performance
mlp_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
mlp_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
mlp_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % mlp_train_accuracy)
print('- MCC: %s' % mlp_train_mcc)
print('- F1 score: %s' % mlp_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % mlp_test_accuracy)
print('- MCC: %s' % mlp_test_mcc)
print('- F1 score: %s' % mlp_test_f1)
Model performance for Training set
- Accuracy: 0.7915964024732997
- MCC: 0.7180212456815344
- F1 score: 0.7923329739707121
----------------------------------
Model performance for Test set
- Accuracy: 0.7855465692668928
- MCC: 0.7099205744424685
- F1 score: 0.7872710924110767
```

In [19]:

```
%%timeit
#Build 2 Stacked model

# Define estimators
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

estimator_list2 = [
    ('knn',knn),
    ('svm_rbf',svm_rbf)]

# Build stack model
stack_model2 = StackingClassifier(
    estimators=estimator_list2, final_estimator=LogisticRegression()
)

# Train stacked model
stack_model2.fit(X_train, y_train)
```

```
# Make predictions
y_train_pred = stack_model2.predict(X_train)
y_test_pred = stack_model2.predict(X_test)

# Training set model performance
stack_model2_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate
Accuracy
stack_model2_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
stack_model2_train_f1 = f1_score(y_train, y_train_pred, average='weighted') #
Calculate F1-score

# Test set model performance
stack_model2_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
stack_model2_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
stack_model2_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate
F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % stack_model2_train_accuracy)
print('- MCC: %s' % stack_model2_train_mcc)
print('- F1 score: %s' % stack_model2_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % stack_model2_test_accuracy)
print('- MCC: %s' % stack_model2_test_mcc)
print('- F1 score: %s' % stack_model2_test_f1)


#Results stacj list 2 estimator_list2
acc_train_list2 ={
'knn':knn_train_accuracy,
'svm_rbf': svm_rbf_train_accuracy,
'stack': stack_model2_train_accuracy}

mcc_train_list2 ={
'knn':knn_train_mcc,
'svm_rbf': svm_rbf_train_mcc,
'stack': stack_model2_train_mcc}

f1_train_list2 ={
'knn':knn_train_f1,
'svm_rbf': svm_rbf_train_f1,
'stack': stack_model2_train_f1}
mcc_train_list2

acc_df2 = pd.DataFrame.from_dict(acc_train_list2, orient='index',
columns=['Accuracy'])
mcc_df2 = pd.DataFrame.from_dict(mcc_train_list2, orient='index', columns=['MCC'])
f1_df2 = pd.DataFrame.from_dict(f1_train_list2, orient='index', columns=['F1'])
df = pd.concat([acc_df2, mcc_df2, f1_df2], axis=1)
round(df, 4).to_csv('stack/T1U1_test_results-65_35-2-Stacked.csv')
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.991849353569421
- MCC: 0.9890997354059421
- F1 score: 0.9918779393372866
----------------------------------
Model performance for Test set
- Accuracy: 0.9686929298199843
```

```
- MCC: 0.9577726283679227
- F1 score: 0.9687831974022592
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.991849353569421
- MCC: 0.9890997354059421
- F1 score: 0.9918779393372866
----------------------------------
Model performance for Test set
- Accuracy: 0.9686929298199843
- MCC: 0.9577726283679227
- F1 score: 0.9687831974022592
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.991849353569421
- MCC: 0.9890997354059421
- F1 score: 0.9918779393372866
----------------------------------
Model performance for Test set
- Accuracy: 0.9686929298199843
- MCC: 0.9577726283679227
- F1 score: 0.9687831974022592
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.991849353569421
- MCC: 0.9890997354059421
- F1 score: 0.9918779393372866
----------------------------------
Model performance for Test set
- Accuracy: 0.9686929298199843
- MCC: 0.9577726283679227
- F1 score: 0.9687831974022592
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Model performance for Training set
- Accuracy: 0.991849353569421
- MCC: 0.9890997354059421
- F1 score: 0.9918779393372866
----------------------------------
Model performance for Test set
- Accuracy: 0.9686929298199843
- MCC: 0.9577726283679227
- F1 score: 0.9687831974022592
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.991849353569421
- MCC: 0.9890997354059421
- F1 score: 0.9918779393372866
----------------------------------
Model performance for Test set
- Accuracy: 0.9686929298199843
- MCC: 0.9577726283679227
- F1 score: 0.9687831974022592
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.991849353569421
- MCC: 0.9890997354059421
- F1 score: 0.9918779393372866
----------------------------------
Model performance for Test set
- Accuracy: 0.9686929298199843
- MCC: 0.9577726283679227
- F1 score: 0.9687831974022592
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.991849353569421
- MCC: 0.9890997354059421
- F1 score: 0.9918779393372866
----------------------------------
Model performance for Test set
- Accuracy: 0.9686929298199843
- MCC: 0.9577726283679227
- F1 score: 0.9687831974022592
34.2 s ± 6.74 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [ ]:

Appendix D: 3-Stack Model for the dataset on Background-Traffic and Bandwidth-Limit Experiment

In [1]:

```
#T1=64 bytes
#T2=58956 bytes
#T3=65507 bytes
#
#U1=1M
#U2=256M
#U3=512M
```

In [2]:

```
#import the neede libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import VotingClassifier
from sklearn.feature_selection import VarianceThreshold

from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import f1_score

import seaborn as sns
import matplotlib.pyplot as plt

# Comment this if the data visualisations doesn't work on your side
%matplotlib inline

plt.style.use('bmh')
```

In [3]:

```
#%%timeit
#load the dataset
#df = pd.read_csv('data-LANS-SDN-INTERNET/T1U1.csv')

#load the dataset
df=pd.read_csv('data-LANS-SDN-INTERNET/T1U2.csv')
#len(df)
df.shape
```

Out[3]:

```
(7772, 15)
```

In [4]:

```
df
```

Out[4]:

7772 rows × 15 columns

In [5]:

```
list(set(df.dtypes.tolist()))
```

Out[5]:

```
[dtype('O'), dtype('int64'), dtype('float64')]
```

In [6]:

```
df_num = df.select_dtypes(include = ['float64', 'int64'])
df_num.head()
df_num
```

Out[6]:

	T1	U2	SW	T	PN	BN	IA	PI	FS	FD
0	64	256	1	4.684	1	91	4	29	0	0
1	64	256	1	4.549	1	141	4	30	0	0
2	64	256	1	7.324	1	42	7	30	1	1
3	64	256	1	7.317	1	42	7	29	1	1
4	64	256	1	5.174	1	42	5	29	1	1
...	...	...	...	...	...	...	...	...	...	...
7767	64	256	8	26.229	99526	150483312	0	29	0	0
7768	64	256	5	11.203	996	2741640	0	29	0	0
7769	64	256	9	26.268	99640	150655680	0	2	0	0
7770	64	256	3	26.435	99741	150808392	0	29	0	0
7771	64	256	2	26.469	99844	150964128	0	12	0	0

7772 rows × 10 columns

In [7]:

```
#using Protocol as target variable
X=df.drop(columns=['PR','ANSD', 'RIFD','ANSS','RIFS'],  axis=1)
y=df['PR']
```

In [8]:

```
X
```

Out[8]:

	T1	U2	SW	T	PN	BN	IA	PI	FS	FD
0	64	256	1	4.684	1	91	4	29	0	0
1	64	256	1	4.549	1	141	4	30	0	0
2	64	256	1	7.324	1	42	7	30	1	1
3	64	256	1	7.317	1	42	7	29	1	1
4	64	256	1	5.174	1	42	5	29	1	1
...	...	...	...	...	...	...	...	...	...	...
7767	64	256	8	26.229	99526	150483312	0	29	0	0
7768	64	256	5	11.203	996	2741640	0	29	0	0
7769	64	256	9	26.268	99640	150655680	0	2	0	0
7770	64	256	3	26.435	99741	150808392	0	29	0	0
7771	64	256	2	26.469	99844	150964128	0	12	0	0

7772 rows × 10 columns

In [9]:

```
selection = VarianceThreshold(threshold=(0.1))
X = selection.fit_transform(X)
X.shape
```

Out[9]:

```
(7772, 8)
```

In [10]:

```
X_train, X_test, y_train, y_test = train_test_split(
X, y, stratify=y, test_size=0.35, random_state=42)

X_train.shape, X_test.shape
```

Out[10]:

```
((5051, 8), (2721, 8))
```

In [11]:
```
y_train.value_counts()
```
Out[11]:
```
tcp     2567
udp     1316
arp      605
icmp     563
Name: PR, dtype: int64
```
In [12]:
```
y_test.value_counts()
```
Out[12]:
```
tcp     1383
udp      709
arp      326
icmp     303
Name: PR, dtype: int64
```
In [13]:
```
#Build Classification models

from sklearn.metrics import accuracy_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import f1_score
```
In [14]:
```
#K nearest neighbors

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(3) # Define classifier
knn.fit(X_train, y_train) # Train model

# Make predictions
y_train_pred = knn.predict(X_train)
y_test_pred = knn.predict(X_test)

# Training set performance
knn_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
knn_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
knn_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate F1-
score


# Test set performance
knn_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
knn_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
knn_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % knn_train_accuracy)
print('- MCC: %s' % knn_train_mcc)
print('- F1 score: %s' % knn_train_f1)
print('---------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % knn_test_accuracy)
print('- MCC: %s' % knn_test_mcc)
print('- F1 score: %s' % knn_test_f1)
Model performance for Training set
- Accuracy: 0.9883191447238171
- MCC: 0.9819578067746086
- F1 score: 0.9883065499196092
---------------------------------
Model performance for Test set
- Accuracy: 0.9687614847482543
- MCC: 0.951660483046525
- F1 score: 0.9686704938760865
```
In [15]:
```
#Support vector machine (Radial basis function kernel)
```

```
from sklearn.svm import SVC

svm_rbf = SVC(gamma=2, C=1)
svm_rbf.fit(X_train, y_train)

# Make predictions
y_train_pred = svm_rbf.predict(X_train)
y_test_pred = svm_rbf.predict(X_test)

# Training set performance
svm_rbf_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
svm_rbf_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
svm_rbf_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate
F1-score

# Test set performance
svm_rbf_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
svm_rbf_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
svm_rbf_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-
score

print('Model performance for Training set')
print('- Accuracy: %s' % svm_rbf_train_accuracy)
print('- MCC: %s' % svm_rbf_train_mcc)
print('- F1 score: %s' % svm_rbf_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % svm_rbf_test_accuracy)
print('- MCC: %s' % svm_rbf_test_mcc)
print('- F1 score: %s' % svm_rbf_test_f1)
Model performance for Training set
- Accuracy: 1.0
- MCC: 1.0
- F1 score: 1.0
----------------------------------
Model performance for Test set
- Accuracy: 0.7140757074604924
- MCC: 0.5748695123002997
- F1 score: 0.6740937529445371
```

In [16]:

```
#Decision tree

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(max_depth=5) # Define classifier
dt.fit(X_train, y_train) # Train model

# Make predictions
y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)

# Training set performance
dt_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
dt_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
dt_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate F1-
score

# Test set performance
dt_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
dt_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
dt_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % dt_train_accuracy)
print('- MCC: %s' % dt_train_mcc)
print('- F1 score: %s' % dt_train_f1)
print('----------------------------------')
```

```
print('Model performance for Test set')
print('- Accuracy: %s' % dt_test_accuracy)
print('- MCC: %s' % dt_test_mcc)
print('- F1 score: %s' % dt_test_f1)
Model performance for Training set
- Accuracy: 0.8515145515739457
- MCC: 0.7762359933845943
- F1 score: 0.8035784366909816
---------------------------------
Model performance for Test set
- Accuracy: 0.8467475192943771
- MCC: 0.7683660880645217
- F1 score: 0.7990145844476247
```

In [17]:

```
#Random forest

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=10) # Define classifier
rf.fit(X_train, y_train) # Train model

# Make predictions
y_train_pred = rf.predict(X_train)
y_test_pred = rf.predict(X_test)

# Training set performance
rf_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
rf_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
rf_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate F1-
score

# Test set performance
rf_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
rf_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
rf_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % rf_train_accuracy)
print('- MCC: %s' % rf_train_mcc)
print('- F1 score: %s' % rf_train_f1)
print('---------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % rf_test_accuracy)
print('- MCC: %s' % rf_test_mcc)
print('- F1 score: %s' % rf_test_f1)
Model performance for Training set
- Accuracy: 0.9986141358146902
- MCC: 0.9978589961133834
- F1 score: 0.9986134911385087
---------------------------------
Model performance for Test set
- Accuracy: 0.9886071297317163
- MCC: 0.9823883753583599
- F1 score: 0.9885909870402152
```

In [18]:

```
#Neural network

from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(alpha=1, max_iter=1000)
mlp.fit(X_train, y_train)

# Make predictions
y_train_pred = mlp.predict(X_train)
y_test_pred = mlp.predict(X_test)

# Training set performance
mlp_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
```

```
mlp_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
mlp_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate F1-
score

# Test set performance
mlp_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
mlp_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
mlp_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % mlp_train_accuracy)
print('- MCC: %s' % mlp_train_mcc)
print('- F1 score: %s' % mlp_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % mlp_test_accuracy)
print('- MCC: %s' % mlp_test_mcc)
print('- F1 score: %s' % mlp_test_f1)
```

```
Model performance for Training set
- Accuracy: 0.6072064937636111
- MCC: 0.3784847549650167
- F1 score: 0.5991816599019475
---------------------------------
Model performance for Test set
- Accuracy: 0.5994119808893789
- MCC: 0.36334178043767823
- F1 score: 0.5923456251587758
```

In [19]:

```
%%timeit
#Build 3 Stacked model

# Define estimators
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

estimator_list = [
    ('knn',knn),
    ('svm_rbf',svm_rbf),
    ('dt',dt)]

# Build stack model
stack_model = StackingClassifier(
    estimators=estimator_list, final_estimator=LogisticRegression()
)

# Train stacked model
stack_model.fit(X_train, y_train)

# Make predictions
y_train_pred = stack_model.predict(X_train)
y_test_pred = stack_model.predict(X_test)

# Training set model performance
stack_model_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate
Accuracy
stack_model_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
stack_model_train_f1 = f1_score(y_train, y_train_pred, average='weighted') #
Calculate F1-score

# Test set model performance
stack_model_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
stack_model_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
stack_model_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate
F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % stack_model_train_accuracy)
print('- MCC: %s' % stack_model_train_mcc)
```

```
print('- F1 score: %s' % stack_model_train_f1)
print('---------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % stack_model_test_accuracy)
print('- MCC: %s' % stack_model_test_mcc)
print('- F1 score: %s' % stack_model_test_f1)


#Results

acc_train_list = {'knn':knn_train_accuracy,
'svm_rbf': svm_rbf_train_accuracy,
'dt': dt_train_accuracy,
'stack': stack_model_train_accuracy}

mcc_train_list = {'knn':knn_train_mcc,
'svm_rbf': svm_rbf_train_mcc,
'dt': dt_train_mcc,
'stack': stack_model_train_mcc}

f1_train_list = {'knn':knn_train_f1,
'svm_rbf': svm_rbf_train_f1,
'dt': dt_train_f1,
'stack': stack_model_train_f1}


mcc_train_list
#import pandas as pd

acc_df = pd.DataFrame.from_dict(acc_train_list, orient='index', columns=['Accuracy'])
mcc_df = pd.DataFrame.from_dict(mcc_train_list, orient='index', columns=['MCC'])
f1_df = pd.DataFrame.from_dict(f1_train_list, orient='index', columns=['F1'])
df = pd.concat([acc_df, mcc_df, f1_df], axis=1)
round(df, 4).to_csv('stack/T1U2_test_results-65_35-3Stacked.csv')
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9978222134230845
- MCC: 0.9966341559761256
- F1 score: 0.9978216067790154
---------------------------------
Model performance for Test set
- Accuracy: 0.9845644983461963
- MCC: 0.9761243874608249
- F1 score: 0.9845448058208809
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9978222134230845
- MCC: 0.9966341559761256
- F1 score: 0.9978216067790154
---------------------------------
Model performance for Test set
```

```
- Accuracy: 0.9845644983461963
- MCC: 0.9761243874608249
- F1 score: 0.9845448058208809
/home/olonlonse/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9978222134230845
- MCC: 0.9966341559761256
- F1 score: 0.9978216067790154
----------------------------------
Model performance for Test set
- Accuracy: 0.9845644983461963
- MCC: 0.9761243874608249
- F1 score: 0.9845448058208809
/home/olonlonse/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9978222134230845
- MCC: 0.9966341559761256
- F1 score: 0.9978216067790154
----------------------------------
Model performance for Test set
- Accuracy: 0.9845644983461963
- MCC: 0.9761243874608249
- F1 score: 0.9845448058208809
/home/olonlonse/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9978222134230845
- MCC: 0.9966341559761256
- F1 score: 0.9978216067790154
----------------------------------
Model performance for Test set
- Accuracy: 0.9845644983461963
- MCC: 0.9761243874608249
- F1 score: 0.9845448058208809
/home/olonlonse/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

```
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9978222134230845
- MCC: 0.9966341559761256
- F1 score: 0.9978216067790154
---------------------------------
Model performance for Test set
- Accuracy: 0.9845644983461963
- MCC: 0.9761243874608249
- F1 score: 0.9845448058208809
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9978222134230845
- MCC: 0.9966341559761256
- F1 score: 0.9978216067790154
---------------------------------
Model performance for Test set
- Accuracy: 0.9845644983461963
- MCC: 0.9761243874608249
- F1 score: 0.9845448058208809
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9978222134230845
- MCC: 0.9966341559761256
- F1 score: 0.9978216067790154
---------------------------------
Model performance for Test set
- Accuracy: 0.9845644983461963
- MCC: 0.9761243874608249
- F1 score: 0.9845448058208809
23.3 s ± 5.15 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [ ]:

Appendix E: 5-Stack Model for the dataset on Background-Traffic and Bandwidth-Limit Experiment

In [1]:

```
#T1=64 bytes
#T2=58956 bytes
#T3=65507 bytes
#
#U1=1M
#U2=256M
#U3=512M
```

In [2]:

```
#import the neede libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import VotingClassifier
from sklearn.feature_selection import VarianceThreshold

from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import f1_score

import seaborn as sns
import matplotlib.pyplot as plt

# Comment this if the data visualisations doesn't work on your side
%matplotlib inline

plt.style.use('bmh')
```

In [3]:

```
#%%timeit
#load the dataset
#df = pd.read_csv('data-LANS-SDN-INTERNET/T1U1.csv')

#load the dataset
df=pd.read_csv('data-LANS-SDN-INTERNET/T1U1.csv')

#len(df)
df.shape
```

Out[3]:

```
(10949, 15)
```

In [4]:
```
df
```
Out[4]:
10949 rows × 15 columns
In [5]:
```
list(set(df.dtypes.tolist()))
```
Out[5]:
```
[dtype('O'), dtype('float64'), dtype('int64')]
```
In [6]:
```
df_num = df.select_dtypes(include = ['float64', 'int64'])
df_num.head()
df_num
```
Out[6]:

	T1	U1	SW	T	PN	BN	IA	PI	FS	FD
0	64	1.0	2	0.008	1	42	0	29	1	1
1	64	1.0	2	0.013	1	42	0	12	1	1
2	64	1.0	3	0.015	1	42	0	24	1	1
3	64	1.0	3	0.015	1	42	0	29	1	1
4	64	1.0	4	0.017	1	42	0	29	1	1
...	...	...	...	...	...	...	...	...	...	...
10944	64	1.0	5	29.276	91986	139082832	0	29	0	0
10945	64	1.0	7	12.760	92	90699	0	29	0	0
10946	64	1.0	6	14.076	922	2444260	0	29	0	0
10947	64	1.0	9	12.762	94	92925	0	2	0	0
10948	64	1.0	8	29.679	94805	143345160	0	29	0	0

10949 rows × 10 columns
In [7]:
```
#using Protocol as target variable
X=df.drop(columns=['PR','ANSD', 'RIFD','ANSS','RIFS'],  axis=1)
y=df['PR']
```
In [8]:
```
X
```
Out[8]:

	T1	U1	SW	T	PN	BN	IA	PI	FS	FD
0	64	1.0	2	0.008	1	42	0	29	1	1
1	64	1.0	2	0.013	1	42	0	12	1	1
2	64	1.0	3	0.015	1	42	0	24	1	1
3	64	1.0	3	0.015	1	42	0	29	1	1
4	64	1.0	4	0.017	1	42	0	29	1	1
...	...	...	...	...	...	...	...	...	...	...
10944	64	1.0	5	29.276	91986	139082832	0	29	0	0
10945	64	1.0	7	12.760	92	90699	0	29	0	0
10946	64	1.0	6	14.076	922	2444260	0	29	0	0
10947	64	1.0	9	12.762	94	92925	0	2	0	0
10948	64	1.0	8	29.679	94805	143345160	0	29	0	0

10949 rows × 10 columns
In [9]:
```
selection = VarianceThreshold(threshold=(0.1))
X = selection.fit_transform(X)
X.shape
```
Out[9]:
```
(10949, 8)
```

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(
X, y, stratify=y, test_size=0.35, random_state=42)

X_train.shape, X_test.shape
```
Out[10]:
```
((7116, 8), (3833, 8))
```
In [11]:
```
y_train.value_counts()
```
Out[11]:
```
tcp     2361
icmp    1647
arp     1646
udp     1462
Name: PR, dtype: int64
```
In [12]:
```
y_test.value_counts()
```
Out[12]:
```
tcp     1272
icmp     888
arp      886
udp      787
Name: PR, dtype: int64
```
In [13]:
```
#Build Classification models

from sklearn.metrics import accuracy_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import f1_score
```
In [14]:
```
#K nearest neighbors

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(3) # Define classifier
knn.fit(X_train, y_train) # Train model

# Make predictions
y_train_pred = knn.predict(X_train)
y_test_pred = knn.predict(X_test)

# Training set performance
knn_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
knn_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
knn_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate F1-
score


# Test set performance
knn_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
knn_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
knn_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % knn_train_accuracy)
print('- MCC: %s' % knn_train_mcc)
print('- F1 score: %s' % knn_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % knn_test_accuracy)
print('- MCC: %s' % knn_test_mcc)
print('- F1 score: %s' % knn_test_f1)
Model performance for Training set
- Accuracy: 0.9799044406970208
- MCC: 0.9728832016332284
- F1 score: 0.9799160124647879
```

```
---------------------------------
Model performance for Test set
- Accuracy: 0.9663448995564832
- MCC: 0.9546482883106525
- F1 score: 0.9663757244554873
```
In [15]:
```
#Support vector machine (Radial basis function kernel)

from sklearn.svm import SVC

svm_rbf = SVC(gamma=2, C=1)
svm_rbf.fit(X_train, y_train)

# Make predictions
y_train_pred = svm_rbf.predict(X_train)
y_test_pred = svm_rbf.predict(X_test)

# Training set performance
svm_rbf_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
svm_rbf_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
svm_rbf_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate
F1-score

# Test set performance
svm_rbf_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
svm_rbf_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
svm_rbf_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-
score

print('Model performance for Training set')
print('- Accuracy: %s' % svm_rbf_train_accuracy)
print('- MCC: %s' % svm_rbf_train_mcc)
print('- F1 score: %s' % svm_rbf_train_f1)
print('---------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % svm_rbf_test_accuracy)
print('- MCC: %s' % svm_rbf_test_mcc)
print('- F1 score: %s' % svm_rbf_test_f1)
Model performance for Training set
- Accuracy: 0.9998594716132658
- MCC: 0.999810290548095
- F1 score: 0.999859480751884
---------------------------------
Model performance for Test set
- Accuracy: 0.7685885729193843
- MCC: 0.7205294173458786
- F1 score: 0.7482163114218551
```
In [16]:
```
#Decision tree

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(max_depth=5) # Define classifier
dt.fit(X_train, y_train) # Train model

# Make predictions
y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)

# Training set performance
dt_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
dt_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
dt_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate F1-
score

# Test set performance
dt_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
dt_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
```

```
dt_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % dt_train_accuracy)
print('- MCC: %s' % dt_train_mcc)
print('- F1 score: %s' % dt_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % dt_test_accuracy)
print('- MCC: %s' % dt_test_mcc)
print('- F1 score: %s' % dt_test_f1)
```

```
Model performance for Training set
- Accuracy: 0.898397976391231
- MCC: 0.8665683088153157
- F1 score: 0.8973465038587761
----------------------------------
Model performance for Test set
- Accuracy: 0.8912079311244456
- MCC: 0.8567556597007966
- F1 score: 0.8899898031351843
```

In [17]:

```
#Random forest

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=10) # Define classifier
rf.fit(X_train, y_train) # Train model

# Make predictions
y_train_pred = rf.predict(X_train)
y_test_pred = rf.predict(X_test)

# Training set performance
rf_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
rf_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
rf_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate F1-
score

# Test set performance
rf_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
rf_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
rf_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % rf_train_accuracy)
print('- MCC: %s' % rf_train_mcc)
print('- F1 score: %s' % rf_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % rf_test_accuracy)
print('- MCC: %s' % rf_test_mcc)
print('- F1 score: %s' % rf_test_f1)
```

```
Model performance for Training set
- Accuracy: 0.9990163012928611
- MCC: 0.9986718700575713
- F1 score: 0.9990163944036905
----------------------------------
Model performance for Test set
- Accuracy: 0.9856509261674928
- MCC: 0.9806263197999506
- F1 score: 0.9856611660445321
```

In [18]:

```
#Neural network

from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(alpha=1, max_iter=1000)
mlp.fit(X_train, y_train)
```

```
# Make predictions
y_train_pred = mlp.predict(X_train)
y_test_pred = mlp.predict(X_test)

# Training set performance
mlp_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accuracy
mlp_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
mlp_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate F1-
score

# Test set performance
mlp_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
mlp_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
mlp_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1-score

print('Model performance for Training set')
print('- Accuracy: %s' % mlp_train_accuracy)
print('- MCC: %s' % mlp_train_mcc)
print('- F1 score: %s' % mlp_train_f1)
print('---------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % mlp_test_accuracy)
print('- MCC: %s' % mlp_test_mcc)
print('- F1 score: %s' % mlp_test_f1)
Model performance for Training set
- Accuracy: 0.8128161888701517
- MCC: 0.7489337123685648
- F1 score: 0.8068914600681564
---------------------------------
Model performance for Test set
- Accuracy: 0.8158100704409079
- MCC: 0.752874360964493
- F1 score: 0.8093013373520986
```

In [19]:

```
%%timeit
#Build 5 Stacked model
# Define estimators
estimator_list = [
    ('knn',knn),
    ('svm_rbf',svm_rbf),
    ('dt',dt),
    ('rf',rf),
    ('mlp',mlp) ]
# Build stack model
stack_model = StackingClassifier(
    estimators=estimator_list, final_estimator=LogisticRegression()
)

# Train stacked model
stack_model.fit(X_train, y_train)

# Make predictions
y_train_pred = stack_model.predict(X_train)
y_test_pred = stack_model.predict(X_test)

# Training set model performance
stack_model_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate
Accuracy
stack_model_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
stack_model_train_f1 = f1_score(y_train, y_train_pred, average='weighted') #
Calculate F1-score

# Test set model performance
stack_model_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
stack_model_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
stack_model_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate
F1-score
```

```
print('Model performance for Training set')
print('- Accuracy: %s' % stack_model_train_accuracy)
print('- MCC: %s' % stack_model_train_mcc)
print('- F1 score: %s' % stack_model_train_f1)
print('---------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % stack_model_test_accuracy)
print('- MCC: %s' % stack_model_test_mcc)
print('- F1 score: %s' % stack_model_test_f1)


#esults

acc_train_list = {'knn':knn_train_accuracy,
'svm_rbf': svm_rbf_train_accuracy,
'dt': dt_train_accuracy,
'rf': rf_train_accuracy,
'mlp': mlp_train_accuracy,
'stack': stack_model_train_accuracy}

mcc_train_list = {'knn':knn_train_mcc,
'svm_rbf': svm_rbf_train_mcc,
'dt': dt_train_mcc,
'rf': rf_train_mcc,
'mlp': mlp_train_mcc,
'stack': stack_model_train_mcc}

f1_train_list = {'knn':knn_train_f1,
'svm_rbf': svm_rbf_train_f1,
'dt': dt_train_f1,
'rf': rf_train_f1,
'mlp': mlp_train_f1,
'stack': stack_model_train_f1}


mcc_train_list
import pandas as pd

acc_df = pd.DataFrame.from_dict(acc_train_list, orient='index', columns=['Accuracy'])
mcc_df = pd.DataFrame.from_dict(mcc_train_list, orient='index', columns=['MCC'])
f1_df = pd.DataFrame.from_dict(f1_train_list, orient='index', columns=['F1'])
df = pd.concat([acc_df, mcc_df, f1_df], axis=1)
round(df, 4).to_csv('stack/T1U1_test_results-65_35-5Stacked.csv')
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9998594716132658
- MCC: 0.999810290548095
- F1 score: 0.999859480751884
---------------------------------
Model performance for Test set
- Accuracy: 0.9903469866944952
- MCC: 0.9869857891319864
- F1 score: 0.9903680369733026
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9997189432265318
- MCC: 0.9996206440562371
- F1 score: 0.9997189797355517
---------------------------------
Model performance for Test set
- Accuracy: 0.9908687711974954
- MCC: 0.9876920041207727
- F1 score: 0.9908885159977375
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9997189432265318
- MCC: 0.9996206440562371
- F1 score: 0.9997189797355517
---------------------------------
Model performance for Test set
- Accuracy: 0.9887816331854944
- MCC: 0.984862070388131
- F1 score: 0.9887982580777016
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9998594716132658
- MCC: 0.999810290548095
- F1 score: 0.999859480751884
---------------------------------
Model performance for Test set
- Accuracy: 0.9887816331854944
- MCC: 0.9848734237030116
- F1 score: 0.9888119594508434
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9997189432265318
- MCC: 0.9996206440562371
- F1 score: 0.9997189797355517
---------------------------------
Model performance for Test set
- Accuracy: 0.9887816331854944
- MCC: 0.9848519013492678
```

```
- F1 score: 0.9887876481363194
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 0.9997189432265318
- MCC: 0.9996206440562371
- F1 score: 0.9997189797355517
----------------------------------
Model performance for Test set
- Accuracy: 0.990086094442995
- MCC: 0.9866211656315284
- F1 score: 0.9901012686905423
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 1.0
- MCC: 1.0
- F1 score: 1.0
----------------------------------
Model performance for Test set
- Accuracy: 0.9924341247064962
- MCC: 0.9897847651947784
- F1 score: 0.9924387090691755
/home/olonlonse/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model performance for Training set
- Accuracy: 1.0
- MCC: 1.0
- F1 score: 1.0
----------------------------------
Model performance for Test set
- Accuracy: 0.9895643099399948
- MCC: 0.9859211148843842
- F1 score: 0.9895847027165893
48.4 s ± 13.7 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Bio-data

A. PERSONAL DATA

Name:	SADIKU ISIAKA BABATUNDE
Address:	No 18, Ligbein Compound, Ode-Remo, Ogun State, Nigeria
Telephone Number:	08030849415
Date / Place of Birth:	7th June, 1972 / Ode Remo, Remo North L. G.
Email:	sibsadiku@gmail.com

B. EDUCATIONAL BACKGROUND

Lead City University, Ibadan (PhD Computer Science)	In View
University of Ilorin (M. Engr. Agricultural and Biosystems Engineering)	2018
Lead City University, Ibadan (M. Sc. Computer Science)	2017
Ladoke Akintola University of Technology Ogbomoso (PGD Computer Science)	2010
Federal University of Technology Akure (B. Engr. Agricultural Engineering)	2000

C. WORK EXPERIENCE

Gateway (ICT) Polytechnic Saapade, Ogun State.	2006 Till date
Ogun State Teaching Service Commission (Orile Isaga Secondary School)	2004-2006
Real Time Technology Gbagada Lagos (Trainee Engineer)	2001-2004

D. MEMBERSHIP OF PROFESSIONAL BODIES

FELLOWSHIP MEMBERSHIP / ASSOCIATE MEMBERSHIP

PROFESSIONAL ASSOCIATION/MEMBERSHIP

1. Fellow, Association of Applied Information Management Professional

2. Member, Linux Professional Institute (LPI)

3. Member, Nigeria Society of Engineers (NSE)

4. Member Society for Conservation Biology (SCB)

5. Registered member Council for Regulation of Engineering (COREN). R23,345

6. Member, Chartered Institute of Local Government and Public Administration of Nigeria

7. Member, Special Marshals Federal Road Safety Corps (FRSC) RS2.24 Ogere

E. NAMES AND ADDRESS OF REFEREES

1. Dr. W. Sakpere

Head of Department

Department of Computer Science

Lead City University, Ibadan, Oyo State, Nigeria

+234 8159582869

2. Dr. S. K. Oseni

Rector

Gateway (ICT) Polytechnic, Saapade, Ogun State, Nigeria

+234 8063155041

_____ _____

Signature Date

The University Compliance Certification

This is to certify that the thesis by Isiaka Babatunde SADIKU in the Department of Computer Science, Faculty of Natural and Applied Sciences, Lead City University, Ibadan is in full compliance with the University format and style.

_____ _____

Signature Date