Automated Machine Learning Driven Quality of Service Management in Resource-constrained Software Defined Networks

Keegan Thomas White

Department of Computer Science

University of Cape Town

Supervisor

Dr Josiah Chavula

In partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

February 2023

Acknowledgements

I want to extend my heartfelt gratitude to my parents for their unwavering support and encouragement throughout my academic journey. Their love and belief in me have been a constant source of motivation and strength. Without them, I would not have been able to achieve a fraction of what I have achieved.

I would also like to express my sincerest thanks to my supervisor, Dr Josiah Chavula, who has been a mentor and guide every step of the way. His expertise and guidance have been invaluable in shaping my research, and this dissertation would not have been possible without his support.

Finally, I would like to thank the Telkom Centre of Excellence (CoE), the Council for Scientific and Industrial Research (CSIR) and iNethi for their support and resources throughout this dissertation.

Abstract

Community networks are a means to bridge the connectivity gaps present in low-income and rural areas. Many of these networks are resource-constrained, mesh-based, and connected to the Internet via low-capacity links. These characteristics result in poor network performance. Software Defined Networking facilitates dynamic resource allocation to address real-time network degradation. Using the Software Defined Networking paradigm, methods to identify what traffic to allocate resources to offer a promising solution to common network issues in community networks. This dissertation presents a novel end-to-end framework that uses deep learning models to facilitate real-time resource allocation in a resource-constrained network based on heuristics for traffic prioritisation. The deep learning models utilised by the framework are trained on data gathered from a community network and extensively tested in online network simulations. The results of this study convey that deep learning enabled Software Defined Networks can improve network throughput and decrease packet loss in real-time, thus improving network Quality of Service.

Contents

List of Figures

List of Tables

Glossary

API Application Programming Interface. 13

CHPC Centre for High Performance Computing. 28

CNN Convolutional Neural Network. 8

CPU Central Processing Unit. 3

CSIR Council for Scientific and Industrial Research. 28

CWN Community Wireless Network. 1

DDR4 Double Data Rate Fourth Generation. 29

DPI Deep Packet Inspection. 12

GB Gigabyte. 28

GPU Graphics Processing Unit. 3

IP Internet Protocol. 7

IPv4 Internet Protocol version 4. 7

kbps kilobits per second. 40

LSTM Long Short-Term Memory. 11

MB megabytes. 36

Mbps megabits per second. 36

NaaS Network-as-a-Service. 21

NIC Network Interface Controller. 39

NN Neural Network. 8

ODL OpenDaylight. 15

ONOS Open Network Operating System. 4

OSI Open Systems Interconnection. 23

PCAPs Packet Captures. 7

PPS Packets Per Second. 70

QoS Quality of Service. 1

RAM Random-Access Memory. 29

RNN Recurrent Neural Network. 8

SDN Software Defined Networking. 1

TCP Transmission Control Protocol. 23

TLS Transport Layer Security. 23

VoIP Voice over Internet Protocol. 6

WAN Wide Area Networks. 15

Chapter 1

Introduction

1.1 Motivation

A Community Wireless Network (CWN) is a decentralised, community-owned and operated network that typically provides community members with access to the Internet and locally-hosted services [1; 2]. However, these networks' wireless links are inherently unpredictable, with asymmetrical link features that cause network degradation due to suboptimal routing configurations and dynamic signal strength [2].

CWNs are becoming more common as Internet access becomes ever more critical, and Internet pricing models remain inequitable. Given the appeal of these networks, they can grow exponentially. Their decentralised nature means self-management tools are imperative to ensure that network Quality of Service (QoS) remains satisfactory.

The Software Defined Networking (SDN) paradigm [3] has emerged in response to the limitations of traditional networking. The core features of an SDN are the separation of the control and data plane, the addition of a controller with a centralised, global network view, and network programmability [4; 5]. These

features allow network administrators to create rules that address QoS in real-time, with various approaches such as inter-domain routing, resource reservation, queue management, and scheduling mechanisms [4].

Converting traditional mesh-based CWNs to use the SDN paradigm will address the typical QoS issues these networks face. However, existing approaches only enable fine-grained traffic control on a flow or application level, where information about these flows is based on packet header information [4]. Using packet header information relies on the network administrators gathering and analysing application flow-level information before being able to put QoS management systems in place. Given the exponential growth in the number of services on the Internet and the variety of services that network clients will use, flow analysis is time-consuming and it has significant limitations [5]. To combat manual flow analysis and the need to gather flow-level information, packet-byte-based machine learning methods for encrypted traffic classification have emerged as a method to generalise classification into categories. This type of classification focuses on using encrypted packet payloads for classification. Packet-byte-based machine learning methods for encrypted traffic classification are hypothesised to be functional in real-time [6; 7; 8]. However, there is currently no existing work implementing this type of traffic classification in real-time. Implementing this type of classification would enable dynamic resource allocation based on traffic categories in an SDN environment.

1.2 Problem Statement

SDN offers resource allocation mechanisms that can be paired with packet-byte-based machine learning methods for real-time encrypted traffic classification to enable the prioritisation of network traffic and dynamic allocation of resources.

This dissertation tests the viability of implementing an SDN-based CWN framework that can classify traffic and allocate resources based on these classifications to improve QoS.

1.3 Research Questions

The following research questions are formulated and answered by this dissertation:

- Are lightweight deep learning models feasible for real-time packet-byte-based encrypted traffic classification in a resource-constrained SDN?
- What overhead is incurred, in terms of network QoS, when adding a real-time packet-byte-based traffic classifier to a resource-constrained SDN?
- Can existing, pre-packaged SDN QoS management tools use real-time traffic classification data to improve network QoS?

1.4 Contribution

This dissertation presents an end-to-end framework that will allow QoS rules to be automatically invoked on traffic flows based on their packets' payloads, where these payloads are fed into a deep learning model that will classify the flow into a category. In this study, the defined categories are streaming, messaging, social media, and peer-to-peer torrenting traffic. The framework is designed and tested for Central Processing Unit (CPU) bound servers rather than Graphics Processing Unit (GPU) bound servers, given that CWNs are usually made up of affordable, low-resource devices. The per-flow classifications this low-resource classifier produces are used to apply per-flow rate limiting that is dynamically allocated to improve network throughput and decrease packet loss.

The data used to train and test these models was collected over four months in the iNethi[1] CWN, with three months of data used for offline training, validation and testing and one month's data used as unseen data for online experiments in a virtual SDN.

To the best of the authors' knowledge, this study is the first attempt at using real-world network data to create an end-to-end traffic classification and QoS management framework that is evaluated in online network simulations.

1.5 Overview

This study is divided into three sections. Firstly, collating and analysing traffic from the iNethi CWN to assist with designing and implementing the classification and QoS management framework and the test environment. Insights on the popular traffic categories, the number of network clients, the size (in bytes) of the packets sent, and the size and length of the flows are gathered. This information influences the traffic used to form categories for the framework, the number of bytes fed into the framework, and the test environments' size and complexity.

The second portion of the study focuses on designing and testing deep learning algorithms to classify encrypted network traffic. Varying algorithm complexity is used with different amounts of input data to form a range of algorithm types that differ in data pre-processing and classification time.

Finally, a virtualised SDN environment is designed using Mininet[2] and the Open Network Operating System (ONOS)[3] SDN controller to carry out online network simulations.

[1] https://www.inethi.org.za/
[2] http://mininet.org/
[3] https://opennetworking.org/onos/

Chapter 2

Background

This study revolves around CWNs, network traffic classification using Deep Learning, and SDNs. These concepts are explored below.

2.1 Community Wireless Networks

Despite growth in smartphone adoption, Internet usage and access to data are limited in South Africa by prohibitive costs and unequal coverage [9; 10]. This has led to the creation of CWNs, such as the iNethi[1] CWN. CWNs are decentralised systems in which an inbound Internet connection is disseminated throughout a community via access points. Community members build and maintain these networks to create a more accessible and affordable Internet connection in areas where infrastructure is absent or insufficient [9; 11]. Infrastructural issues are endemic in low-income and rural areas as there is no economic motivation, in terms of profitability, for telecommunications companies to invest in these areas [11].

There are two general CWN categories, wireless mesh and hotspot-based ar-

[1] https://www.inethi.org.za/

chitectures, with prominent CWNs implemented as wireless mesh networks [12]. Ubiquiti's mesh solution[1], often referred to as UniFi Mesh, employs a multi-hop relay system. This means that access points can relay data for other access points, expanding network coverage without the need for a wired backbone. They utilize a proprietary protocol to ensure efficient routing and minimize potential interference, making network deployment flexible and scalable. The iNethi CWN utilises UniFi Mesh. CWNs tend to be operated by non-profit organisations in collaboration with local stakeholders. They are resource-constrained and lack a budget for the latest technology, and infrastructure [9; 11].

A broad range of applications are used in CWNs, such as Voice over Internet Protocol (VoIP), content distribution, on-demand and live media streaming, instant messaging, remote backups and updates, file storage, and file sharing [11]. These services are restricted in CWNs in terms of QoS by the limited capacity of servers and wireless links, the structure and diameter of the network graph, and fluctuations in the network due to load and faults. Studies into the usage of these applications show that community members tend to communicate and engage with those within their locality [9; 13]. For this reason, locally-hosted services are also a standard offering in CWNs.

The iNethi network, depicted in Figure 2.1, is an example of a community-owned non-profit CWN. The data utilised in this study is gathered from the iNethi CWN. This network is a wireless mesh network located in a low-income community named Ocean View in Cape Town, South Africa [10]. It provides Internet access and access to locally-hosted content and services across the community via wireless access points [9]. Mesh access points were chosen during the construction of the iNethi network as they offer self-setup and self-healing when there are link issues [9]. While iNethi facilitates Internet connection, its focus is

[1]https://help.ui.com/hc/en-us/articles/115002262328-UniFi-Network-Considerations-for-Optimal-Wireless-Mesh-Networks

Figure 2.1: Location of the Mesh Access Points of the iNethi Network in Ocean View, Cape Town

providing localised alternatives to Internet-based applications due to the market identified in their communities [9; 10]. Services are hosted within the communities where iNethi is located and are accessible on the same mesh network that provides Internet access.

To gauge community interest in the network and monitor network usage, Packet Captures (PCAPs) are recorded hourly at the Internet gateway using tcpdump[1]. These PCAP files contain all layer two network information for Internet traffic originating within the network from the previous hour, including source and destination Internet Protocol (IP) addresses, source and destination port numbers and Internet Protocol version 4 (IPv4) payloads. The data gathered from the iNethi network is analysed in Section 5.1.

[1]https://www.tcpdump.org/

2.2 Deep Learning

Deep learning algorithms consist of a multiple Neural Network (NN) architecture [14]. A NN is a biologically inspired model consisting of an input layer, an optional hidden or many hidden layers and an output layer that extracts linear combinations from inputs as derived features and then models the target as a nonlinear function of these features [14; 15]. Generally, the feature extraction process is done within the hidden layers. These layers consist of a single or, more regularly, multiple nodes that apply an activation function to the previous layer's output [14]. This is eventually fed to the output layer, which performs the classification or regression by applying an activation function. Since feature extraction and the classification or regression task are generally done by an extensive sequence of hidden layers with interconnected nodes, these models are hard to analyse and understand.

The Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) architectures are currently the most accurate and feasible deep learning algorithms to use in real-time traffic classification [6; 7; 8].

CNNs have shown excellent results in traffic classification tasks due to their ability to handle multidimensional data. The core concept behind CNNs is their use of convolutional layers, which can automatically and adaptively learn spatial hierarchies or patterns from data. In the context of network traffic classification, CNNs can effectively capture the local dependencies and structural information of network traffic data, identifying unique patterns associated with various types of network traffic.

RNNs, on the other hand, have an inherent capability to deal with sequential data. They are designed to remember previous inputs in the sequence using hidden layers. This makes them particularly suitable for time-series data, such as network traffic, where temporal dependencies play a crucial role. In real-time

traffic classification, the sequence of data packets and the temporal correlation between them are leveraged by the RNN to achieve accurate classification results.

2.2.1 Convolutional Neural Networks

CNNs were first introduced by LeCun et al. [16] as a multilayer NN model known as LeNet-5. LeNet-5 was designed to classify handwritten digits. While LeNet-5 does not perform well on modern-day, complex problems, this can be attributed to a lack of computing power at the time of conception. This lack of computing power in conjunction with a lack of large-scale training datasets are the main contributors to poor performance compared to the state-of-the-art adaptations of LeNet-5 [17].

CNNs can be implemented with various architectures. However, their basic architecture, depicted in Figure 2.2, is generally very similar [17]. These models have three layers: a convolutional, pooling and fully-connected layer. The convolutional layer learns feature representations by creating feature maps convolved with various learned kernels, with the results being passed into a nonlinear activation function [17]. This produces complete feature maps that reduce the number of parameters [14; 17].

The pooling layer aims to reduce the resolution of the feature map, thus reducing the feature size [14]. These layers are usually placed between convolutional layers. A pooling feature map is linked to its respective convolutional feature map, and thus there is the same number of feature maps in both layers. These pooling actions are carried out by various functions such as average or max pooling [14; 18].

The fully connected layer and the output layer serve as the decision-making components of the model. The fully connected layer, after receiving the output from the preceding convolutional and pooling layers, processes high-level features

to learn non-linear combinations of them. These features are then passed to the output layer, which creates a probability distribution over the classes. Essentially, the fully connected layer aggregates learned features and uses them to classify the input, while the Output layer provides the final classification prediction.

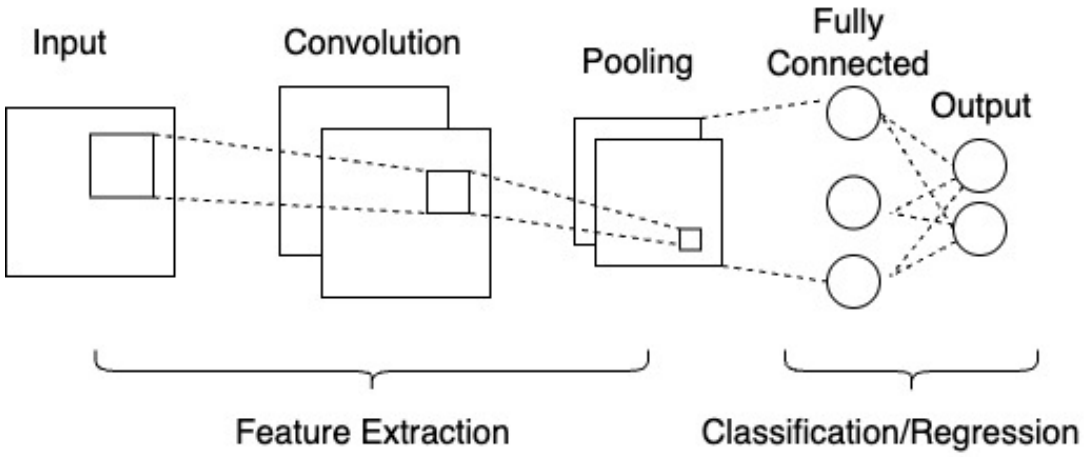

Figure 2.2: Basic CNN Architecture

CNNs are more efficient than traditional NNs as they reuse and share weights rather than kernel weights only being used once [19]. This reduces both storage and memory requirements. Additionally, there has been a significant increase in the popularity of CNNs, leading to major innovations in computer vision, natural language processing and other niche areas. However, most of these models are too resource-intensive for real-time deployment [19].

2.2.2 Recurrent Neural Networks

RNNs are designed to deal with sequential or time-dependent data [20]. The network has loops that preserve inputs using internal memory [14]. This allows the network to make decisions based on the current input and rules it has learnt from previous inputs. It back propagates errors through layers to learn recurrently.

A standard RNN is a nonlinear dynamic system that maps sequences to sequences in a deep feed-forward NN with a layer for each timestep [21]. RNNs most popular implementation is a Long Short-Term Memory (LSTM) model [14]. The LSTM model is an RNN architecture that addresses the vanishing gradient problem using memory units [21]. These units are used to control the gradient that flows between units. Memory units are linear with a self-connection strength of one and a pair of auxiliary gating units that control the input and output of the unit [21]. When these gating units are shut, gradients can flow through them indefinitely with no alteration, thus solving the vanishing gradient problem.

2.2.3 Attention

While RNNs, LSTM NNs and gated-RNNs have been long-established favourites for sequence modelling, encoder-decoder architecture such as attention-based models allow sequence modelling between inputs, no matter the distance between them [22]. This allows for faster processing as attention encoders can be parallelised due to this shift away from the sequential nature of RNNs [8; 22].

An attention function is the mapping of a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. This output is computed as a weighted sum of the values. The weight assigned to each value is computed by a compatibility function of the query with the corresponding key [8; 22]. This can be done in parallel by projecting the query, keys and values into different sub-spaces using learned linear projections and running the attention function on each layer independently. This yields multidimensional results concatenated, projected once again and linearly combined to form output vectors [8; 22].

2.3 Traffic Classification

Network traffic classification is the process of identifying and categorising the type of traffic flowing through a network based on various characteristics such as IP addresses, port numbers, or packet payloads [5; 7; 8; 23]. This can be used for various purposes, such as prioritising different types of traffic to improve network QoS, monitoring for malicious activity, and enforcing network security policies.

There are three main traffic classification methods: Deep Packet Inspection (DPI), port mapping, and machine learning-based classification [5; 8]. With encrypted network traffic becoming commonplace, DPI-based methods are no longer adequate for real-time classification as there is no simple and efficient technique to decrypt traffic payloads and analyse them [5; 7; 8; 23]. Instead, DPI methods use complex natural language processing and are suitable for offline processing [5; 7; 8; 23]. Similarly, port-based methods for traffic classification are also no longer effective due to the increase in dynamic port mapping [5; 7; 8].

This has led most researchers to focus on machine learning-based approaches [8]. These techniques can be divided into flow-feature and packet-byte-based methods [8]. However, in terms of real-time classification, packet-byte-based approaches are more commonly used, as flow-feature-based approaches require a large portion of the whole flow or computation of inter-packet statistics to classify traffic. Another common trait of flow-level classification is the use of protocols, IP addresses and other flow information that makes the model less generalisable. In contrast, packet-byte-based methods only take a short, predetermined number of packets from a flow to classify traffic [6].

Packet-byte-based deep learning models have achieved promising results in offline traffic classification and, while untested, are hypothesised to perform efficiently in real-world scenarios [5]. Additionally, packet-byte-based methods using deep learning have the additional benefit of not requiring manual feature extrac-

tion, which is necessary for many flow-feature-based approaches [24]. Creating an end-to-end framework with this approach does not require expert feature analysis to ensure high accuracy.

To implement a real-time network management system, the traffic classification model must be efficient to ensure timely QoS management. Therefore, many existing offline classification models are not appropriate as they sacrifice efficiency for complexity to achieve high accuracy [6]. In addition, CWNs, such as the iNethi network, are often resource-constrained, meaning the classifier needs to be efficient enough to perform well while limited by the processing power available.

2.4 Software Defined Networking

In this section, the core components of SDNs are explored. An explanation of the key architectural features of SDNs is detailed, along with the protocols and technologies that make up an SDN stack. The core components of the SDN paradigm that differentiate it from traditional networking architecture are also identified and explained.

2.4.1 Network Architecture

The SDN paradigm is built on the notion of separating the control plane and the data plane, as seen in Figure 2.3. This allows the network to have a centralised control mechanism, the SDN controller in Figure 2.3, that can monitor and measure active network traffic to compute network metrics and shift between network policies [14]. Additionally, the network becomes programmable by shifting away from static, manual operations to a fully configurable and dynamic entity.

In an SDN, the routers and switches are forwarding devices that the SDN controller controls via an Application Programming Interface (API). The switches

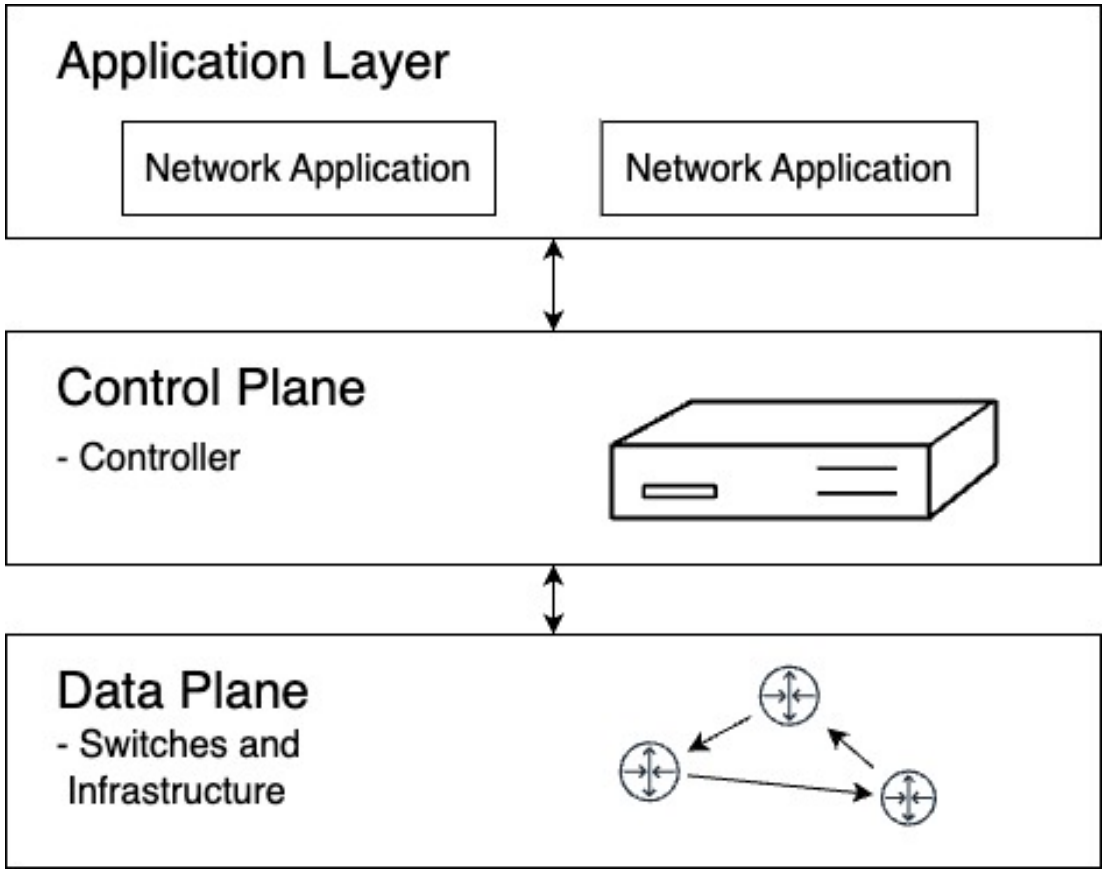

Figure 2.3: SDN Architecture

and routers are also programmable and can notify the controller of specific traffic types, such as elephant flows [25].

Additionally, SDN architecture can be extended to facilitate machine learning-based traffic monitoring [26]. However, utilising the controller in such a way can become a bottleneck when trying to scale the network [14; 27]. Distributed multi-controller platforms [28] have been proposed to address this issue. A distributed multi-controller platform consists of one logically centralised root controller and several local controllers [25]. The central controller has a global view and total accessibility to any switch in the network. In contrast, local controllers only have access to and a view of the switches in their domain. However, the implementation

of a multi-controller network does not solve every issue. The central controller is still a single point of failure. This makes controller reliability an essential aspect of network construction. To solve this issue, a cluster of controllers can be deployed to manage the network [25].

Given that the controller is the fundamental element used to manage all operations of the data plane, the performance and capabilities of the controller itself are essential in achieving optimal network performance and the desired functionality. There are numerous popular SDN controllers, such as NOX, POX, Floodlight, OpenDaylight (ODL), ONOS, and Ryu [29; 30]. The aforementioned centralization of the controller as a bottleneck in the SDN paradigm is a serious challenge from the reliability and performance perspective. Some controllers, such as NOX, POX, Floodlight and Ryu, are designed to be run as centralised controllers, whereas ONOS and ODL are designed to be run as distributed controllers [29]. When analysing the application domain of the decentralised controllers, ONOS and ODL, ONOS is more diverse and is designed to control Wide Area Networks (WAN) and multiple other applications. In contrast, ODL is datacenter specific [29].

2.4.2 Quality of Service

QoS is typically defined as the ability of a network to provide the required services for selected network traffic [4]. The primary goal of QoS is to provide priority with respect to QoS parameters, including but not limited to bandwidth, delay, jitter and packet loss [4].

The SDN paradigm emerged to combat the limitations of traditional networking, allowing for complex and advanced QoS frameworks to be implemented in academia and industry. While many solutions to solve QoS limitations have been proposed in these fields, many of them either failed or have not been im-

plemented [4]. The majority of research into SDN QoS management focuses on intra-domain routing problems [4]. The lack of active, meaningful implementations of QoS management solutions is due to the area of focus of this research and real-world limitations that academia has failed to overcome. While single-domain solutions to QoS are meaningful, inter-domain or traffic classification-based solutions would be more widely applicable as Internet connections rely on encrypted communication between various hosts in different networks [4]. Additionally, the logically centralised nature of SDNs creates scalability issues as all QoS-related signalling messages and statistics are sent from the data plane to the controller causing significant overhead [4]. Thus, a shift of focus from intra-domain research to Internet-based research and studies focusing on SDN scalability would improve the viability of SDN-based QoS management frameworks.

The diverse categories of applications on the Internet all require specific bandwidth and resource allocations to guarantee satisfactory QoS. Traditional 'best-effort' delivery models are not capable of serving the diverse array of traffic flows, while existing SDN and traditional QoS frameworks have not been successful at solving these issues [4]. However, by the very nature of the SDN paradigm, the tools to address network QoS needs are built into the network architecture. The separation of the control and data plane with a logically centralised controller provides network programmability that enables dynamic per-flow resource reservation and routing control mechanisms, thus facilitating dynamic flow optimisation to address real-time network degradation [4; 31].

The OpenFlow Protocol [32] is the most prevalent communication protocol used in SDNs [4]. There are built-in, fine-grained flow management mechanisms that make OpenFlow a powerful tool to combat network degradation in real time. These OpenFlow characteristics are detailed in Section 2.4.3.

2.4.3 Flow Tables and Meters in OpenFlow

A flow table, seen in Table 2.1, consists of flow entries that match a flow to a set of values that include instructions [31; 32]. These instructions can modify the flow's actions. An example action modification would be diverting a flow to a meter entry. Meters enable the controller to implement various simple QoS operations, such as rate-limiting and can be combined with per-port queues to implement complex QoS frameworks [32].

Table 2.1: Main components of a flow entry in a flow table

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Table 2.1 can be broken down as follows:

- **match fields**: to match against packets. These consist of the ingress port, packet headers, and, optionally, metadata specified by a previous table.
- **priority**: matching precedence of the flow entry.
- **counters**: updates the number for matching packets.
- **instructions**: to modify the action set or pipeline processing.
- **timeouts**: maximum amount of time or idle time before the flow expires.
- **cookie**: opaque data value that the controller chooses. The controller may use this to filter flow statistics, modification and deletion, but it is not used when processing packets.

2.4.4 Mininet

Mininet is the most popular SDN emulation tool [33]. It is an open-source network emulator that allows users to create virtual networks using an SDN controller and

Linux libraries [34]. It creates virtual networks by setting up hosts, switches, and links that can emulate a real network. The Linux networking stack is used to create these network devices, and forwarding planes [33; 34]. In Mininet networks, like real-world SDNs, the controller is used as the mechanism for the control plane to interact with the data plane through the OpenFlow protocol, allowing for the programmatic control of the virtual network.

Mininet allows the user to test complex topologies with repeatable regression tests. However, it cannot exceed the CPU resources, or bandwidth available on the physical machine [33; 34]. This means that if the number of events exceeds the number of threads available in the test machine, the events will not run concurrently [33]. Therefore, to test large networks, emulation must be run on a high-resource machine to ensure concurrency and that no CPU bottlenecks affect experimental results.

2.5 Summary and Conclusions

CWNs in low-income and rural areas are inherently restricted in terms of QoS by their low-cost architecture and mesh links. The SDN paradigm has emerged with various mechanism to address the dynamic QoS issues faced in networks such as CWNs. However, there is a lack of SDN-based research that is applicable to CWNs. The use of real-time encrypted traffic classification in conjunction with the QoS control mechanisms built into the OpenFlow protocol offer promising solutions to address QoS in these networks.

Chapter 3

Related Work

This chapter explores the work used as a framework for the design and implementation of this study. The two key concepts that this study revolves around are SDN-based QoS management frameworks and machine-learning-based traffic classification. Work in these areas is detailed in this chapter.

3.1 SDN Quality of Service Monitoring

QoS can be monitored and managed through built-in mechanisms within the OpenFlow protocol. Researchers have used these QoS tools in various studies and extended them to create complex frameworks that enable traffic engineering. These QoS frameworks rely on communication between the switches and the controller to gather QoS statistics and feed them to external applications that subsequently communicate with the controller to make traffic engineering decisions.

There are limited built-in mechanisms to monitor QoS in OpenFlow-enabled SDNs [31; 46]. SDN switches are simple forwarding devices with limited functionality beyond this. These switches are capable of gathering port-based statistics

such as the number of bits sent and received as well as the number of packets sent and received. The SDN controller can poll these statistics from SDN switches using OpenFlow messages [32]. By using these port-based statistics, it is possible to calculate packet loss, link utilisation and port throughput [31]. Calculating packet loss, link utilisation, and throughput are the most common and trivial approaches to gauge QoS in SDNs without creating custom frameworks or extending pre-existing OpenFlow functionality [31; 46].

Port-based QoS monitoring is commonly used in SDN studies that create applications for traffic engineering processes, such as resource allocation, route selection and resource reservation. Santos et al. [45] gather QoS statistics using OpenFlow messages sent periodically from the controller to the switches. Santos et al. [45] then calculate link utilization and packet loss and make it available to applications outside the SDN architecture. In doing so they feed information to a traffic engineering application that selects the optimal route for traffic according to its flow requirements. This application then communicates with the controller to update the network's state. Similarly, Siniarski et al. [47] use the built-in functionality of OpenFlow switches to gather purpose-built QoS statistics for traffic engineering. The authors of this study use intra-domain packet loss to gauge VoIP quality and provide this data in real-time to network administrators so that traffic engineering can take place.

3.2 Quality of Service Resource Reservation

Resource reservation mechanisms are commonly used in addressing real-time network QoS. Resource reservation mechanisms typically use flow classification and rate-shaping to meet a QoS policy. Traditionally, the flow classification uses packet header fields to classify the packet [4]. Researchers have taken various ap-

proaches to extend the built-in functionality of SDNs to enable QoS management heuristics to be put in place by network administrators.

Kim et al. [40] extend a controller and design an API that creates network slices for different applications and subsequently provides these slices with predefined network resources. These resources are controlled with rate limiting, and queue mapping is used to cope with bandwidth and delay allocation. In doing so, Kim et al. [40] create a QoS control framework that automatically and flexibly programs a network of devices with the necessary QoS parameters to meet predefined heuristics created by network administrators. Bueno et al. [41], and Duan et al. [42] make use of Network-as-a-Service (NaaS) to create QoS monitoring and reservation modules. By gathering QoS statistics, these studies produce end-to-end, autonomous frameworks that provide fine-grained control of flows and dynamically detect and enforce bandwidth requirements in a similar way to Kim et al. [40].

On a smaller scale, per-flow application-based QoS allocation in home networks is explored by Seddiki et al. [43; 44]. Seddiki et al. [43; 44] create a framework designed for low-resource devices that are typical of a home network and a system that is usable by the average, unskilled home user. This system, FlowQoS, allows users to specify the high-level applications that should have high priority, such as VoIP or Adaptive Video Streaming services. Subsequently, real-time application identification takes place for each flow and rules are installed in the data plane by the FlowQos controller. This allows the home router to forward individual flows according to user-specified priorities.

While the studies mentioned above classify traffic for QoS management, they only use packet-header information meaning their classification schemes are manually defined. This results in a classification process that is not generalisable beyond a manual grouping of packet header information for specific applications.

Additionally, these studies show that QoS management is possible using built-in OpenFlow functionality.

3.3 Real-time Traffic Classification

There is little work applying deep-learning methods to create end-to-end frameworks capable of efficient real-time classification. Cheng et al. [6] address this problem by building upon various deep learning-based traffic classification models focusing on efficiency. Cheng et al. [6] note that Zou et al. [8] achieve high classification accuracy by taking advantage of time series relationships between sequential packets. This is achieved with an RNN. However, this only allows two packets to interact with each other in a flow creating sequential dependencies in the classification process. By incorporating a multihead attention module, Zou et al. [8] take advantage of these relationships and remove sequential dependencies as a single packet can interact with all sample packets in a single step, thus reducing the parameter and input size. Additionally, the packet is projected into different subspaces meaning various interactions can be processed in parallel. Their model also includes a CNN that was first proven to extract packet-level features accurately by Wang et al. [7].

While these studies [6; 7; 8] achieved high accuracy, they ran their tests on the lab-generated ISCX VPN-nonVPN traffic dataset [38] rather than data gathered from a real-world network, except for Cheng et al. [6]. Cheng et al. [6] uses a real-world dataset and the ISCX VPN-nonVPN traffic dataset. The ISCX VPN-nonVPN dataset [39], published by the University of New Brunswick, is a commonly used dataset in prominent research papers in the field of encrypted traffic classification [6; 7; 8]. Although Cheng et al. [6] use a real-world dataset, they only sample 1000 flows compared to the ISCX dataset with multiple traffic

categories with over 30 000 flows.

It is unclear what the state-of-the-art models' classification accuracy is, as overall classification accuracy is not addressed in the studies mentioned above except for Wang et al. [7], who stated they achieved an accuracy of 85.8%. Zou et al. [8] and Cheng et al. [6] do not clearly state their models' overall performance across all test cases.

When designing a classification framework, choosing the number of packets for sampling and the number of bytes to feed into the model is vital in packet-byte-based approaches, as these parameters decide the balance between efficiency and accuracy. Zou et al. [8] randomly select three packets in a flow and 784 bytes from each packet, Cheng et al. [6] use the first 144 bytes from each packet but do not stipulate the number of packets they use. However, Cheng et al. [6] do state they start sampling after the three-way Transmission Control Protocol (TCP) handshake. So it can be assumed they sample from the Transport Layer Security (TLS) handshake and onwards. Wang et al. [7] sample 784 bytes from packets but focus on flow-feature-based approaches.

Many existing studies focusing on traffic classification include flow information such as source and destination IP addresses and source and destination port numbers [8], with Wang et al. choosing to include Open Systems Interconnection (OSI) layer information in their training data. However, this expanded feature set can render their models susceptible to overfitting, as they may overly rely on application-specific characteristics when making classifications, thus limiting the generalizability of their findings beyond their sample data.

3.4 Summary and Conclusions

From the works mentioned above, it is clear that deep learning models can classify encrypted Internet traffic, but this approach has not been implemented in an online environment. Additionally, there are cases in which the data used to train these models have not been formatted correctly. Thus the results of these studies are not necessarily a fair representation of the capabilities of deep-learning traffic classification. However, using an efficient deep learning model to classify traffic and feed these classifications to a QoS framework would allow for traffic engineering to take place on a traffic category level rather than an application specific level. Thus, creating a well-balanced dataset and focusing on model efficiency and subsequently utilising pre-existing SDN QoS management mechanisms would allow for the creation of a framework capable of real-time QoS management.

Chapter 4

Methodology

This study is broken up into three sections that use results from the previous section to guide the experimental process. All code is available on GitHub[1]. Firstly, the iNethi CWN traffic logs are labelled and analysed to get an overview of the usage patterns and a guideline for the machine learning design and implementation, which is the second section. The machine learning section involves rigorous testing and experimentation with three machine learning architectures. Single-layer, multi-layer and attention-based one-dimensional CNNs are trialled in offline tests to determine the most effective approach to classify encrypted Internet traffic. Lastly, the models built in the offline machine learning section are trialled in online, simulated network environments. This phase examines the models' accuracy and the effectiveness of the QoS framework's decisions.

4.1 Experimental Overview

The three categories of experimental results align with different research questions and produce results that can provide empirical evidence to answer these questions.

[1] https://github.com/keeganwhite/whtkee004-masters

Offline testing of deep-learning models allows this study to evaluate the effectiveness of lightweight CNNs at classifying encrypted traffic. This enables conclusions to be made about the viability of encrypted traffic classification and to what degree it can be achieved with simple, efficient models. Online experiments evaluate the overhead added to a network when using a classifier and to what degree a classifier can influence network QoS positively or negatively using existing QoS tools built into the SDN paradigm.

While testing whether this approach is viable, it is important to identify the controlled, independent and dependent variables. This allows the proposed research questions to be answered fairly and objectively. The controlled variables in this study are as follows:

- The network from which the traffic is collected.
- The dataset used to create the training and testing datasets.
- The random search parameters.
- The cross-validation degree and number of iterations for offline training.
- The network architectures that are used for online tests.
- The link speed used between hosts in the online tests.
- The traffic replay speed.
- The machine used for testing and training machine learning models.

The independent and dependent variable pairs are as follows:

Table 4.1: Independent and Dependant variables

Independent	Offline Dependant	Online Dependant
Input complexity	Accuracy, F1 Score	Accuracy, pre-processing time
Design Complexity	Accuracy, F1 Score	Network QoS
Hyperparameters	Accuracy, F1 Score	Network QoS

Network QoS referenced in Table 4.1 is measured by gathering port-based statistics from OpenFlow switches in the SDN test environment and calculating the average flow throughput and packet loss.

4.2 Dataset Construction

The dataset used to train the machine learning models in this study comprises real-world traffic flows from the iNethi CWN. PCAP files were captured at the Internet gateway using tcpdump[1] from June to September 2020. These PCAP files are split into flows using pkt2flow[2]. This is done by splitting PCAP files into folders based on their source IP address and port and their destination IP address and port. Both TCP and UDP traffic is included in the dataset.

Following the splitting of the traffic into flows, nDPI[3], a DPI library, is used to label the flows. Only elephant flows are included in the final dataset, where elephant flows are long-lasting traffic flows that require significant network resources [35]. Messaging applications such as WhatsApp and Facebook Messenger are included as they are popular applications that can create long-lasting flows.

With the initial dataset complete, the final training dataset is created by removing flows that do not have enough packets to be regarded as meaningful. For a flow to be meaningful, it needs to have a minimum of seven packets that can be used for classification. This means TCP flows need to have a minimum of fourteen packets, three for the TCP handshake, four for the TLS handshake and seven to use as input into the model. UDP flows require seven or more due to their lack of handshakes.

The final dataset is balanced to ensure models are not biased by one applica-

[1]https://www.tcpdump.org/
[2]https://github.com/caesar0301/pkt2flow
[3]https://www.ntop.org/products/deep-packet-inspection/ndpi/

tion. Each application's traffic is equal to the other applications in their category. The distribution of the dataset can be seen in Table 4.2 with a total of 23 600 flows per category, including traffic from TikTok, YouTube, Instagram, Facebook, BitTorrent, Facebook Messenger and WhatsApp. Although BitTorrent makes up the entire torrenting category and is thus application-specific, it is included as it is the most popular traffic category and the only torrenting traffic found with nDPI.

Table 4.2: Data Distribution of the iNethi CWN Dataset

Traffic Category	Application Name(s)	Total Number of Flows
Video Streaming	TikTok and YouTube	23 600
Social Media	Instagram and Facebook	23 600
Messaging	Facebook Messenger and WhatsApp	23 600
P2P	BitTorrent	23 600

4.3 Machine Learning Model Training and Testing

Three model types are trained and tested in this study. These are a simple single-layer, complex multi-layer and attention-based CNN. These models are trained using the Council for Scientific and Industrial Research (CSIR)[1] Centre for High Performance Computing (CHPC)[2] GPU cluster. The GPU cluster includes nine GPU compute nodes with thirty, twelve-Gigabyte (GB) Nvidia V100[3] GPUs. For training, a single V100 GPU is used with an enforced timeout limit of twelve hours, put in place by CSIR. A CPU-bound machine is used to test the models' accuracy on unseen data. This machine has a Ryzen 5900X CPU[4], with twelve

[1]https://www.csir.co.za/
[2]https://www.chpc.ac.za/
[3]https://www.nvidia.com/en-gb/data-center/tesla-v100/
[4]https://www.amd.com/en/products/cpu/amd-ryzen-9-5900x

cores and twenty-four threads, and 32GB of Double Data Rate Fourth Generation (DDR4) Random-Access Memory (RAM).

The experiments detailed in this section are designed to gauge the viability of the previously mentioned models in terms of their ability to classify encrypted Internet traffic. The key metric to analyse at the end of the experiments detailed below is the classification accuracy achieved by the model on unseen data.

These models are built using the Python-based deep-learning API Keras[1] with a Tensorflow[2] backend. Each model's optimal parameters are chosen through a 200-sample, five-fold cross-validated random search. Based on the F1 score, the top three performing models are grouped, and their hyperparameters form the grid for a five-fold cross-validated grid search. The F1 score, seen in equation 4.1, is used as it is a good representation of the model's generalisability as it is the harmonic mean of the model's precision and recall.

$$\frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{4.1}$$

** TP = True positive, FP = False Positive and FN = False Negative*

At the end of this, the top performing model, based on F1 score, is tested on unseen data where a final accuracy score is calculated as in Equation 4.2.

$$\frac{TP}{Total} \tag{4.2}$$

The dataset is divided into an eighty-twenty split where 80% of the data is used for the random search, grid search, and final training, while 20% of the data is used as unseen data for the final testing phase of the models.

For each model, the input is varied in the same way with the first x bytes of the IPv4 payload sampled from n packets, where x is 144, 196 or 225 bytes.

[1] https://keras.io/
[2] https://www.tensorflow.org/

The choice of x is guided by related work and the structure of the iNethi CWN dataset. One hundred forty-four bytes is chosen based on the work done by Cheng et al. [6], while 225 bytes is selected as the modal class of IPv4 payload length in the iNethi CWN is between 200 and 220 bytes as seen in Figure 4.1. One hundred ninety-six bytes is chosen as an intermediary value between 144 and 225 bytes. When x bytes are sampled from a packet's IPv4 payload, the first twenty bytes contain TCP or UDP header information (where twelve zeros are used to complement the UDP header for alignment with the twenty byte TCP header). These twenty bytes are masked to avoid the models' learning from packet-header information. If a payload does not contain x bytes, it is padded with 0s to have a length of x. Finally, all payload information is in a hexadecimal format, which is converted to decimal data and divided by 255 so that the bytes are converted to values between 0 and 1. This decimal data is saved into a NumPy[1] array and used as the input into the models.

The number of bytes is carefully chosen to keep all inputs transferable to $\sqrt{x}$ by $\sqrt{x}$ arrays that can be converted to grey-scale images where the byte values are divided by 255. This ensures these models follow the standard of equal dimensional inputs set by related work [6; 7; 8].

The number of packets sampled per flow, n, starts at three as per work by Zou et al. [8] and increases by two to five and then seven. These packets are sequential to allow for fast sample times in real-time applications while also allowing the models to build relationships between the sequential nature of Internet packets. However, all inputs are fed into the CNN at once in a x by n array, where each row consists of a single packet. This reduces the complexity of reshaping the packet data gathered while allowing relationships between packets to be explored by grouping inputs in sequential order.

[1] https://numpy.org/

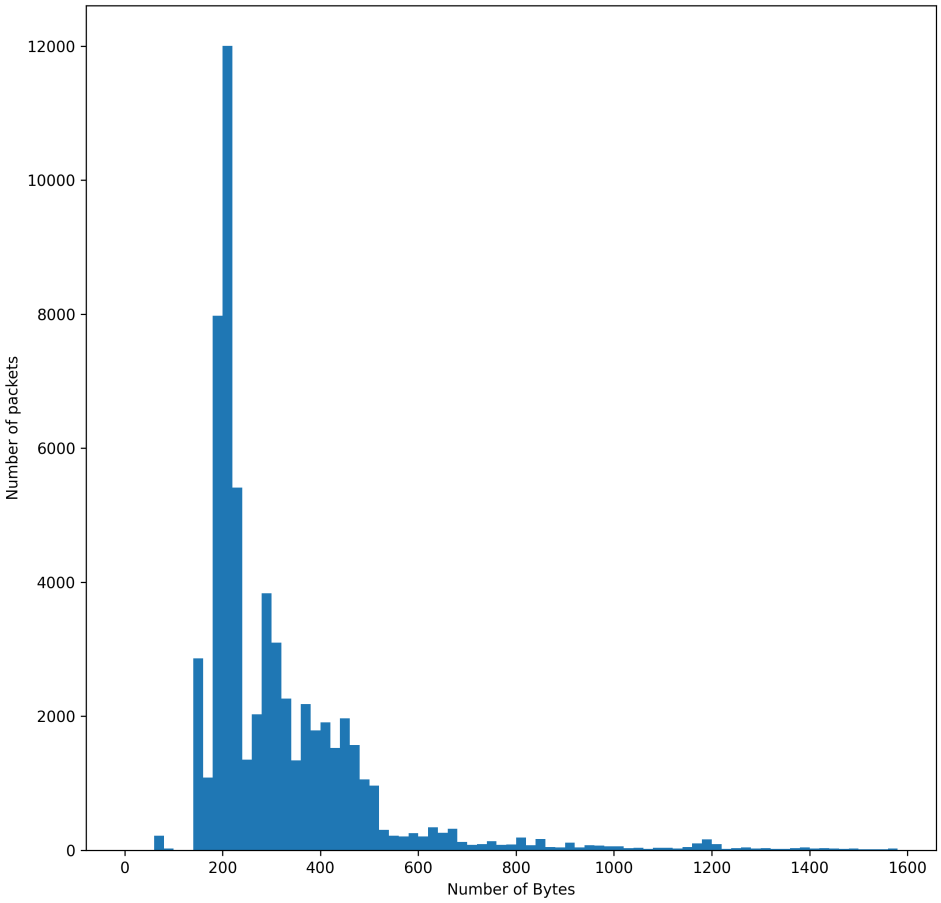

Figure 4.1: IPv4 Payload Length in iNethi CWN Dataset

In summary, the input to every model is a x by n array where each value is between zero exclusively and one inclusively.

4.3.1 Single Layer CNN

Given that CNNs have been proven to be effective at classifying encrypted network traffic [6; 7; 8], a simple, one-dimensional, single-layer CNN is used as a

baseline for both classification time and accuracy.

This CNN consists of an input layer, a 1D convolutional layer, a fully connected layer and an output layer. The full range of hyper-parameters used during the random search process is shown for each layer in Table 4.3.

Table 4.3: Hyperparameters per layer

Layer Name	Hyper-parameters
Batch Size	250, 500, 750, 1000, 2000
Epochs	50, 75, 100, 150, 200, 300, 500
Optimizers	RMSprop, Adam
Filter Sizes	14, 72, 98, 135, 144, 196, 225
Kernel Sizes	1, 3, 5, 7
Hidden Layer Activation Functions	Leaky ReLu, ReLu
Output Layer Activation Functions	Softmax, Sigmoid

4.3.2 Multi-layer CNN

A multi-layer CNN is designed to balance efficiency and accuracy. The basic structure of a CNN is used, as explained in Section 2.2.1, with two hidden layers and a max pooling layer, as shown in Figure 4.2.

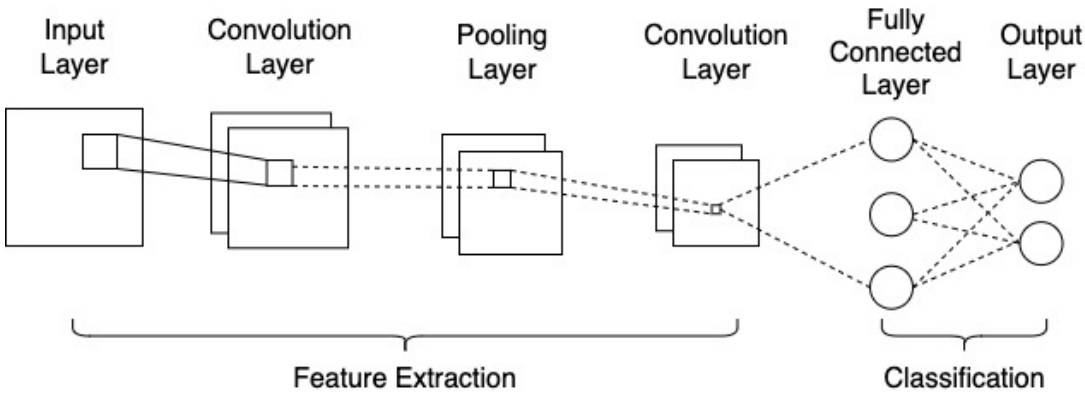

Figure 4.2: Architecture of the multi-layer CNN

The full range of hyperparameters used during the random search process is

shown for each layer in Table 4.4.

Table 4.4: Hyper-parameters per layer

Layer Name	Hyper-parameters
Batch Size	250, 500, 750, 1000, 2000
Epochs	50, 75, 100, 150, 200, 300, 500
Optimizers	RMSprop, Adam
Filter Sizes	14, 72, 98, 135, 144, 196, 225
Kernel Sizes	1, 3, 5, 7
Pool Sizes	1, 3, 5, 7, 10
Dropout Rate	0.02, 0.05, 0.1, 0.2
Hidden Layer Activation Functions	Leaky ReLu, ReLu
Output Layer Activation Functions	Softmax, Sigmoid

4.3.3 Attention-based CNN

Finally, an attention-based model motivated by the work of Zou et al. [8] is designed using a multihead attention approach as explained in Section 2.2.3. This model comprises an input layer, a convolutional layer, a multidimensional attention head, a normalisation layer, a convolutional layer, a fully connected layer and an output layer, as seen in Figure 4.3.

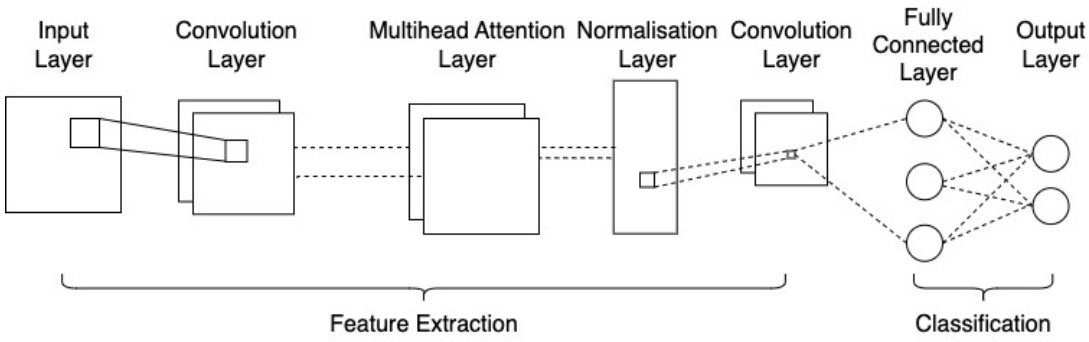

Figure 4.3: Architecture of the Attention-based CNN

The full range of hyperparameters used during the random search process is shown for each layer in Table 4.5.

Table 4.5: Hyper-parameters per layer

Layer Name	Hyper-parameters
Batch Size	250, 500, 750, 1000, 2000
Epochs	50, 75, 100, 150, 200, 300, 500
Optimizers	RMSprop, Adam
Filter Sizes	14, 72, 98, 135, 144, 196, 225
Kernel Sizes	1, 3, 5, 7
Number of Heads	1, 3, 5, 7
Hidden Layer Activation Functions	Leaky ReLu, ReLu
Output Layer Activation Functions	Softmax, Sigmoid

4.4 SDN Test Environment

The SDN test environment uses Mininet to create the simulation topologies natively on the host machine running Ubuntu 22 Desktop LTS. The experiments in this study are run on a Ryzen 5900X CPU, with twelve cores and twenty-four threads. The experiments detailed below are designed to take the most accurate models produced using the experimental process outlined in Section 4.3 and test the overhead they add to network simulations and their influence on network QoS, whether it be positive or negative.

4.4.1 Network Design

The topologies used vary in complexity to test the scalability of the traffic classifier. ONOS is used as the controller, given that it is designed for WAN applications. The Internet gateway, labelled as 'Firewall Switch' if Figure 4.4, has a flow entry that forwards all packets destined for the Internet to the traffic classifier. This means that the classifier and the controller should receive the packets with a negligible difference in arrival time. Thus, it can be stated that the flow classification process starts when the first packet is sent from the client to the controller. Additionally, it is important to note the controller treats the flow in a

standard manner, and a flow entry is made so that the users' experience will not be interrupted by the classification process. The flow entry made by the controller is overwritten by the classifier when a classification for the flow is made.

When receiving a packet, the classifier will group packets into flows based on a four-tuple of source IP address, destination IP address, source port and destination port. Once n packets of this flow are gathered, x bytes are pre-processed and fed into the CNN as per the processes detailed in Section 4.3. The input layer will therefore have an input dimension of *(None, 1, n, x)*. This four-tuple is then marked as used and any other packets in this flow are ignored. This ensures that only the first n packets of the flow are used and there are no duplicate classifications for a particular flow.

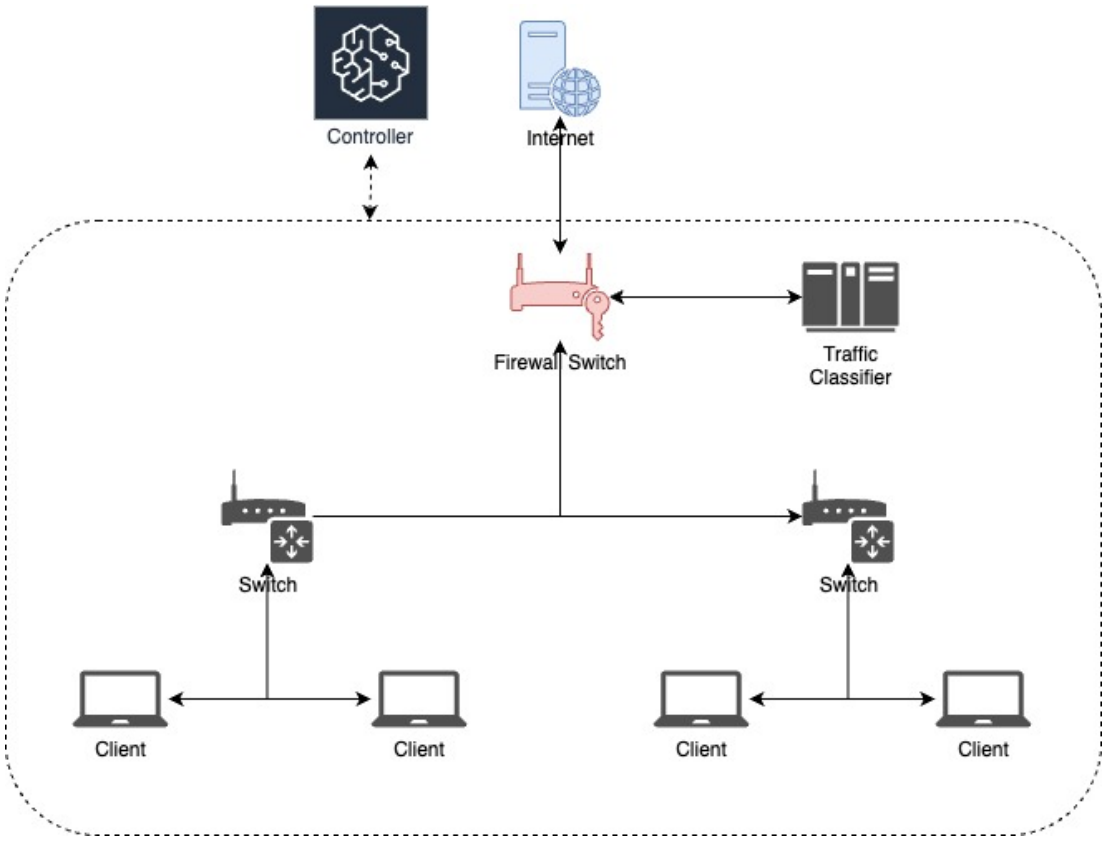

Figure 4.4: An example Tree Topology using the Traffic Classifier

Once a prediction is made, the classifier makes an API call to the controller and installs a flow entry in all switches directing the flow to use the appropriate meter entry. Entries are made in all switches connected to clients because in a mesh network, users can move between access points seamlessly without an interruption to their traffic flow. During this process the time taken to collect the n packets, process the packets and form the CNN's input, make the prediction, make the API call and receive a response as well as the process as a whole is recorded. This allows in-depth analysis of the network and classifier's performance for the different models and network complexities. This analysis can be used to outline the pre-processing time that is incurred before QoS can be affected by a classification.

The links between switches and host devices, as well as switch-to-switch links are limited in Mininet to two megabits per second (Mbps). This ensures that the simulation environment does not waste unnecessary CPU resources in areas that will not affect the accuracy and speed of the aforementioned measurements and is influenced by the fact that the iNethi backhaul speed during data collection was 2mbps.

There are two classes of experiments that are run:

1. Network simulations for each model category to gauge the accuracy of the machine learning models on unseen data in real-time.
2. Network simulations for each network architecture comparing the use of the most accurate classifier compared to a network with no classifier to observe the difference in network QoS.

The above experiments are each conducted with fifteen network clients. The choice for the number of clients is guided by the fact that analysis of the experimental dataset shows fifteen users used over 200 megabytes (MB) of data for applications of interest in the month period when the data was gathered. The

200 MB threshold is chosen as the minimum amount of data used on the network for the client to be deemed significant, as this amount of data could give clients reasonable access to the application categories this study focuses on, especially applications in the messaging category or low-quality short videos in the streaming category [10]. In the experiments, fifteen clients cannot be exceeded as the CPU would become a bottleneck with its 24-thread capacity and affect the ability of the simulation to run switches and clients concurrently.

These experiments are run in a tree topology, starting with one switch, then two and finally with four switches connected to clients. All of these topologies have an external firewall switch which is used as this is the topology of the iNethi CWN. The topologies can be seen in Figures 4.4, 4.5 and 4.6.

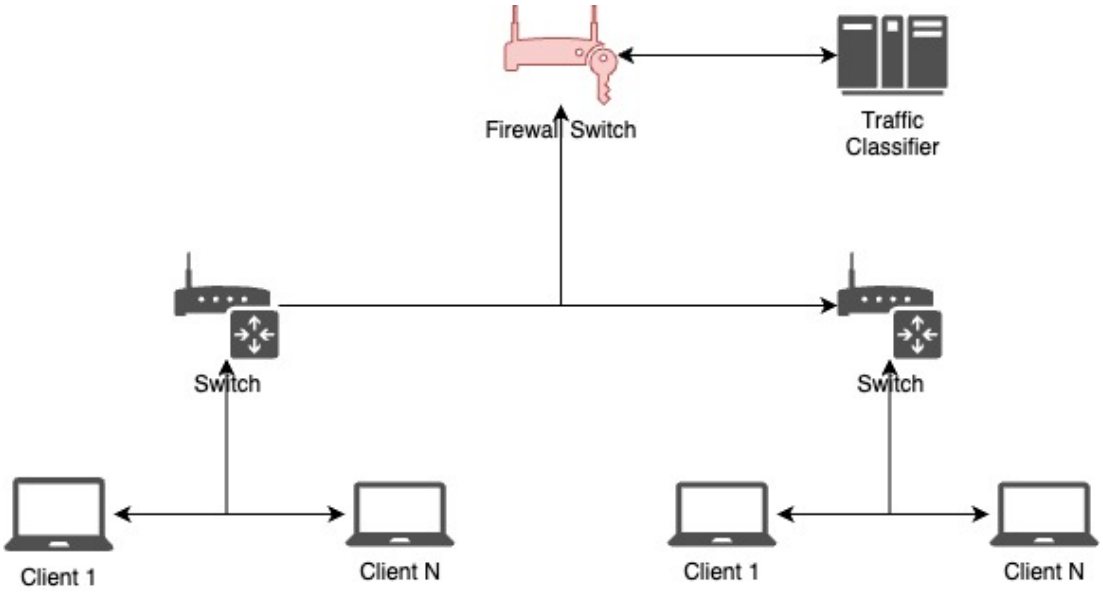

Figure 4.5: Double Switch Test Architecture

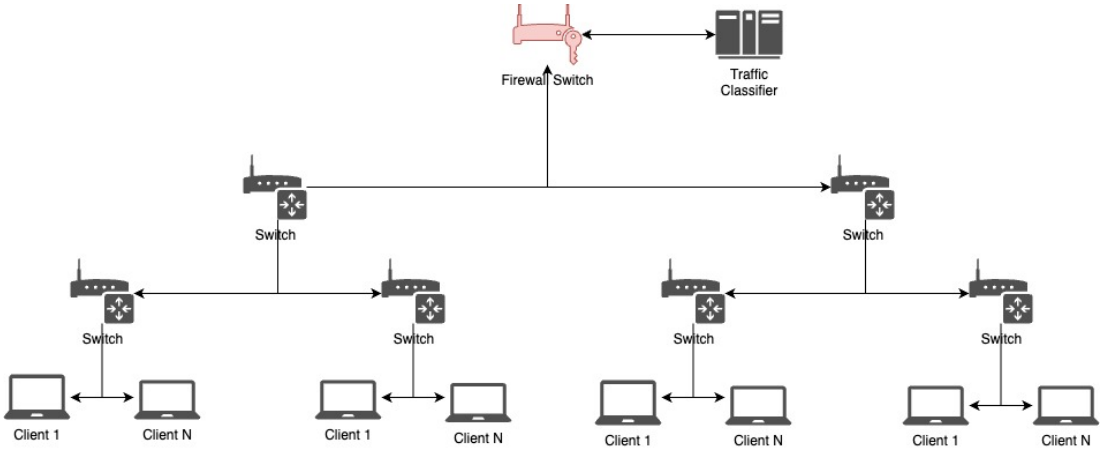

Figure 4.6: Four Switch Test Architecture

4.4.2 Data Selection and Traffic Replay

Each experiment is run for ten minutes, with each client having its own unique collection of UDP and TCP flows. These flows are all members of one of the four categories of traffic. Enough traffic is allocated for each host to run for at least fifteen minutes at 2mbps, ensuring that every client has enough traffic to replay for the entire duration of the experiment. The Scapy[1] Python library is used for traffic replay as it allows simple, repeatable regression tests to be created and run. Scapy uses the opensource Tcpreplay[2] utility to replay traffic. Tcpreplay's basic utility is a resend function that resends all packets from a PCAP file at the speed at which they were captured or at a specified data rate. Due to the fact that it is not a dynamic function, a dropped packet in a TCP traffic replay will not result in re-transmission. Instead, the next packet in the capture file will be replayed. This can result in high packet loss rates when rate-limiting is applied to experiments, whereas, in real-world networks, a re-transmission would take place, and thus packet loss would be significantly reduced. The static nature

[1]https://scapy.net/
[2]https://tcpreplay.appneta.com/

of the replays also mean that UDP packets will be dropped when the switch's capacity is exceeded, rather than the host's replay stalling or slowing down. This will result in high rates of packet loss.

The flow files that are replayed are pre-processed to ensure the source MAC address, and IP address match the client that is replaying the traffic. This is done using the Scapy library. Files in the test dataset are evenly distributed amongst the fifteen potential clients, and once they are allocated to a client, each PCAP file has its source IP address and MAC address changed to match the client's source IP address and MAC address.

Each flow is replayed at the max capacity of the link at 2mbps. To accurately measure throughput and replay traffic, all traffic is replayed in one direction, from the client to the Internet destination. This means that throughput and packet loss can be calculated in a similar manner to related works [36; 37] as will be described in Section 4.4.4.

4.4.3 Data Processing and Traffic Classifier

The traffic classifier runs on a host connected to the Internet gateway switch, known as the 'firewall' in mesh architectures such as the iNethi CWN. The Network Interface Controller (NIC) is monitored using the Scapy Python library to process inbound packets in real time. These packets are analysed with their IPv4 payloads grouped into a Python dictionary based on a four-tuple of their source IP address, destination IP address, source port and destination port. Once an entry into this dictionary has n packets to use as input for the machine learning model, they are pre-processed and then passed to the model. The pre-processing involves manipulating the packets into a Numpy array with the padding and masking operations carried out as detailed in Section 4.3.

Once the classifier makes a prediction, an API call to the ONOS controller

is used to assign this flow to a pre-defined meter entry installed in all switches. There are four meter entries installed in all switches when the network is started, corresponding to the four categories of traffic. These meter entries can be overridden at any point enabling dynamic resource allocation. These meter entries use a drop function to act as rate-limiters to ensure a specific flow never exceeds a specific rate. This allows network administrators to decide and alter resource allocation for traffic categories in real time. For example, a meter entry is defined as follows, it has an assigned ID, a lifetime, speed and function type. The ID is used when assigning a flow to a meter. For instance, if meter ID one is used for the messaging category, then an API call is made to assign message category flows to ID one. The lifetime is the time for which a meter entry is installed in a switch. This lifetime can be infinite or a time in seconds. The speed, in kilobits per second (kbps), is the max rate a flow can transmit at when it is assigned to a meter. Finally, the function used in this study is a drop function, where flows exceeding the max flow rate will have packets dropped to lower their transmission rate. The corresponding flow entry for each flow filters flows based on their source MAC address, ensuring that the classification is linked to the client device. This information is gathered using Scapy during the pre-processing phase.

4.4.4 Quality of Service Monitoring

Average flow and packet loss per switch in the SDN test environment is calculated and recorded every second. These metrics are gathered using API calls built-in to the ONOS SDN controller.

Throughput is calculated as per related work [36] by making an API call and reading all flow entries in the gateway switch. The number of bytes processed during each flow is divided by the number of seconds this flow has been running for and appended to a dictionary key value pair indexed by the MAC address of

the host device.

Packet loss is calculated as per related work [36; 37] by making an API call to each switch every second and gathering port statistics. Given the known topology of the test environment, the number of packets sent by the port connected to the next switch or to the Internet is recorded as well as the number of packets received by the rest of the ports on the device. The packet loss percentage for this device is then calculated by subtracting the received packets from the sent packets and dividing it by the number received packets. Each value is appended to a dictionary key value pair indexed by the switch number. The average of these values is then calculated per switch at the end of the experiment. This process is carried out with the most accurate model, in terms of classification accuracy on unseen data, with rate limits of 100, 200, 200 and 1000 kbps for Torrenting, Messaging, Social Media and Streaming, respectively. This ensures streaming traffic is reserved the most resources. This follows the fact that the majority of Internet traffic is video-based [31]. It is important to note that the rate limits chosen are arbitrary, and the only restraints in speed is the link speed within the network and the backhaul speed.

Chapter 5

Results

This study produces three categories of results, namely, an examination of the iNethi CWN traffic dataset, an examination of the classification accuracy of single-layer, multi-layer and attention-based one-dimensional CNNs in an offline environment, and finally, an analysis of the effectiveness of the aforementioned models on influencing the QoS of an SDN-enabled CWN in real-time.

The results from the dataset analysis influence the choices made in the offline machine learning and online network simulation phases. The results produced during the offline machine learning and online test phases answer the proposed research questions. Specifically, the offline machine learning phase of this study focuses on the viability of using machine learning to classify encrypted Internet traffic. While the online testing phase examines the extent to which machine learning can influence QoS in an SDN and how this affects the network in terms of overhead.

5.1 The iNethi Datasets

There are two separate datasets created from traffic gathered from June through September 2020. The training dataset is made up of three months of traffic and the testing dataset is made up of a month of traffic. This section presents, firstly, the analysis of the original unfiltered iNethi dataset and, secondly, the filtered dataset that is sampled for training and testing

5.1.1 Machine Learning Training Dataset

Figure 5.1 presents the number of filtered and unfiltered flows in the training dataset, where filtering refers to removing flows that do not contain enough packets (less than 14 for TCP flows and less than 7 for UDP flows) to be used in the training process, as detailed in Section 4.2.

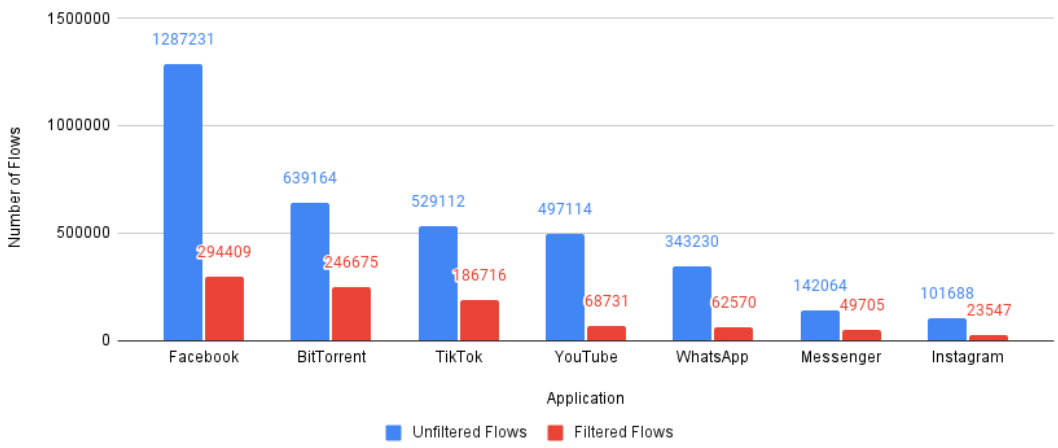

Figure 5.1: Number of Flows per Category for the iNethi CWN Training Dataset

From Figure 5.1, it can be seen that filtering the training dataset results in significantly fewer flows per category. The number of flows is reduced by 77.13%, 61.41%, 64.71%, 68.62%, 81.77%, 96.50% and 76.84% for Facebook, BitTorrent, TikTok, YouTube, WhatsApp, Messenger and Instagram respectively.

Thus, there is a large difference between significant and total flows.

Furthermore, Figure 5.2 shows that the average size of flows is small in the unfiltered training dataset, with only two categories averaging over a megabit in size. These categories are YouTube and Instagram, with 4.88 and 1.41 megabits, respectively. After filtering the dataset, the significant flows of each category have average flow sizes that are about double that of the original dataset, with only the WhatsApp and Messenger categories averaging flow sizes less than a megabit.

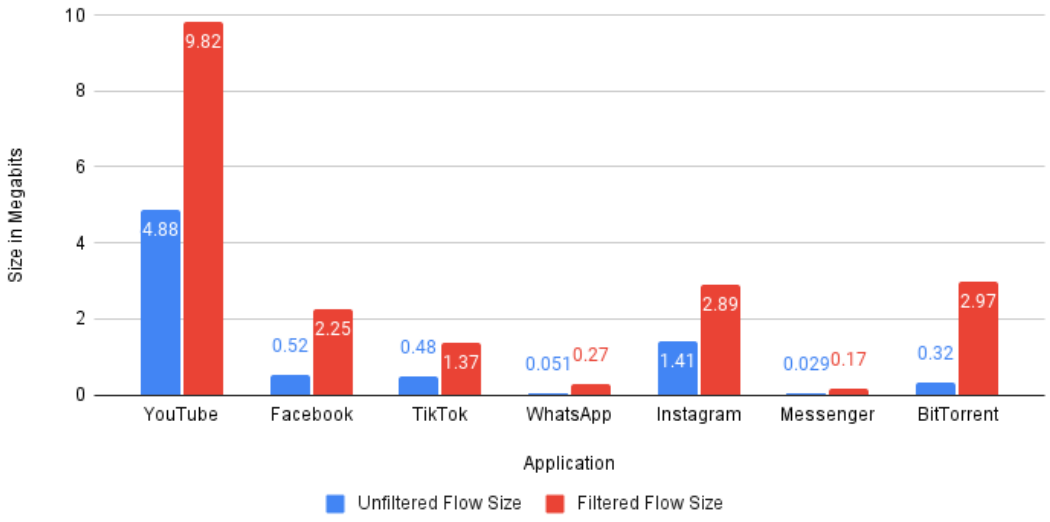

Figure 5.2: Flow Size per Category in the iNethi CWN Training Dataset

5.1.2 Machine Learning Online Testing Dataset

The online testing dataset is sampled from traffic collected in September 2020. The breakdown of the number of flows is present in Figure 5.3. The number of flows is significantly decreased when filtering is done, resulting in Instagram and Facebook Messenger having only 38 and 83 usable flows, respectively. There is a decrease of 72.63%, 84.31%, 92.03%, 95.71%, 96.82%, 98.87% and 91.73% for the YouTube, Facebook, TikTok, WhatsApp, Instagram, Messenger, and BitTorrent

flows, respectively.

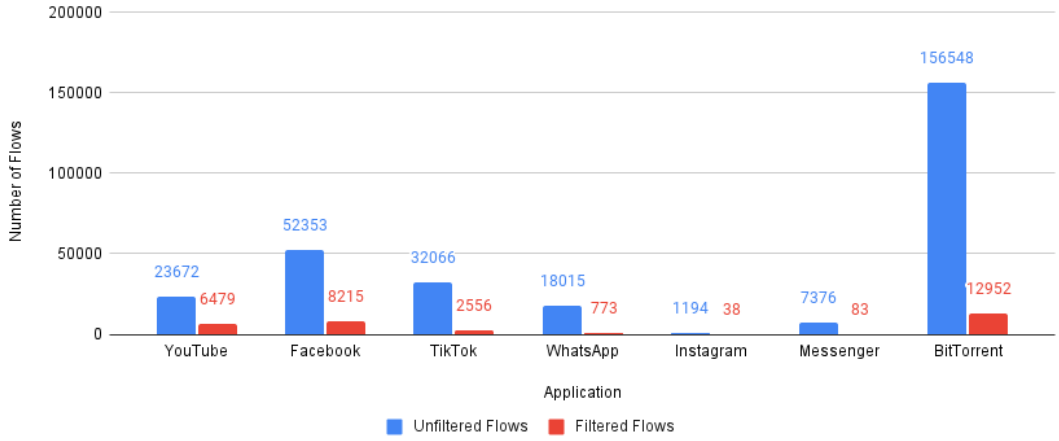

Figure 5.3: Number of Flows per Category in the iNethi CWN Testing Dataset

The average flow size for the testing dataset is presented in Figure 5.4. Only the YouTube category averages over a megabit for the unfiltered dataset with a size of 4.01 megabits. In the filtered dataset there are two categories averaging over a megabit, which are YouTube and Facebook, with a size of 5.69 and 1.06 megabits, respectively.

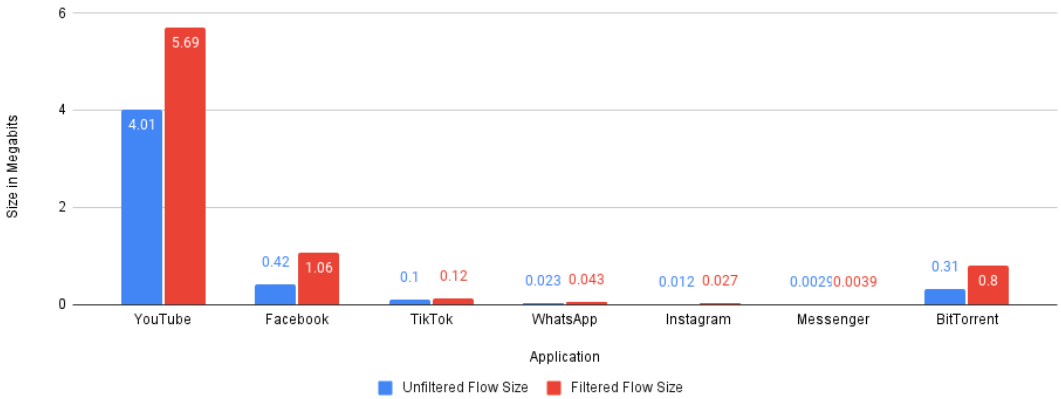

Figure 5.4: Flow Size per Category in the iNethi CWN Online Testing Dataset

5.2 Machine Learning Training and Offline Testing

Four metrics are recorded during the offline deep learning models' development, namely, the model's training accuracy, validation accuracy, F1 score from the grid search process and finally, the model's offline accuracy on unseen data. The models' accuracy on unseen data allows conclusions to be made about the feasibility of using these models to classify encrypted Internet traffic.

5.2.1 Single Layer CNN

As per the results recorded in Table 5.1, increasing the number of bytes and packets sampled per flow increases the single-layer model's accuracy and F1 score. This is true up until the final model in the table which samples 225 bytes from seven packets. This model is 0.12% less accurate on unseen data than the model sampling 225 bytes from five packets which is the most accurate model on unseen data. The confusion matrix of the model sampling 225 bytes from five packets can be seen in Figure 5.5.

Table 5.1: Offline Experimental Results for One-dimensional CNNs

Bytes	Packets	Training Accuracy	Validation Accuracy	F1 Score	Unseen Accuracy
144	3	91.89	85.66	85.89	83.61
144	5	90.85	87.09	86.59	84.32
144	7	91.18	86.84	87.46	86.03
196	3	90.51	86.02	87.51	84.22
196	5	91.61	89.69	88.06	84.85
196	7	93.50	89.03	89.07	85.34
225	3	93.76	87.27	87.89	85.54
225	5	94.35	87.58	89.24	86.24
225	7	97.06	89.08	89.41	86.12

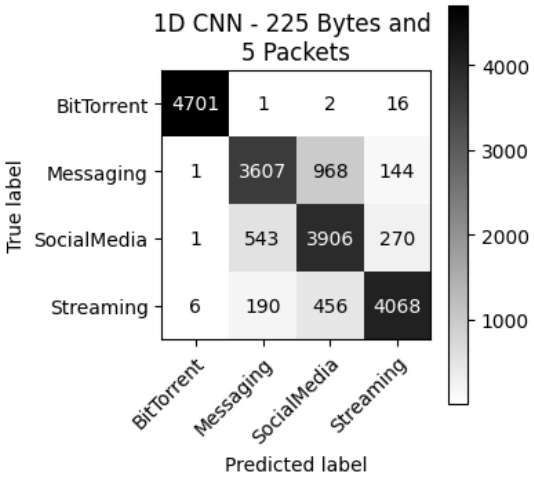

Figure 5.5: Confusion Matrix of Single-layer CNN on Unseen Data

The single-layer CNN sampling five packets and 225 bytes has an overall accuracy on unseen data of 86.24%. This is broken into categories in Figure 5.6. Due to the application-specific nature of the Torrenting category, the accuracy is 13.37% higher than the streaming category, which has the second-highest accuracy. Without Torrenting the average accuracy of the three other categories is 81.80% compared to the previous total of 86.14%.

5.2.2 Multi-layer CNN

As per the results recorded in Table 5.2, there is no direct correlation between increasing the sampled packets and bytes and the F1 score or unseen data accuracy for the multi-layer model category. The best-performing model in terms of F1 score and accuracy on unseen data samples 225 bytes from five packets. The confusion matrix for this model classifying unseen data can be seen in Figure 5.7.

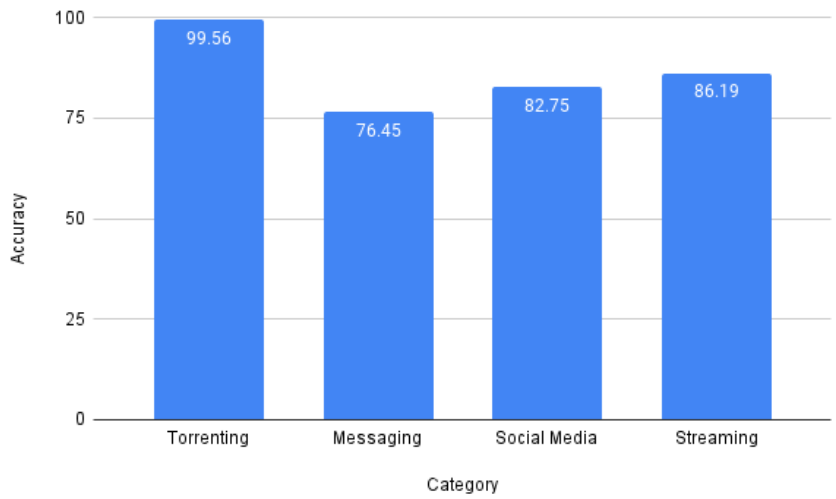

Figure 5.6: Highest Performing Single-layer Model's Accuracy per Category

Table 5.2: Offline Experimental Results for Multi-layer One-dimensional CNNs

Bytes	Packets	Training Accuracy	Validation Accuracy	F1 Score	Unseen Accuracy
144	3	91.65	90.85	92.14	88.37
144	5	93.33	89.39	90.35	87.17
144	7	93.51	90.06	90.78	89.34
196	3	91.71	88.72	91.12	89.77
196	5	92.21	89.96	91.58	89.71
196	7	93.47	90.03	92.28	90.23
225	3	94.28	90.37	93.03	90.34
225	5	94.77	92.46	93.33	90.59
225	7	92.21	88.98	90.71	88.02

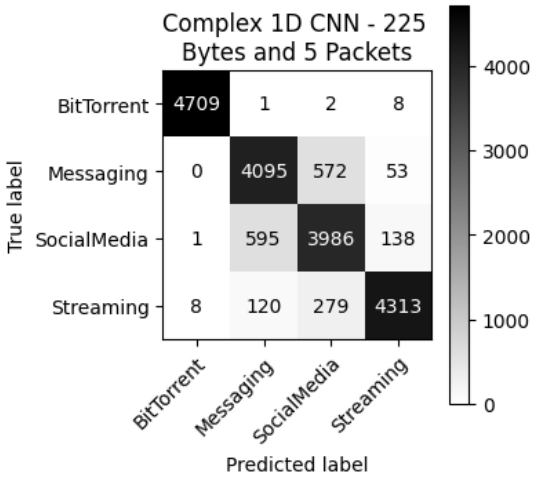

Figure 5.7: Confusion Matrix of Most Accurate Multi-layer CNN on Unseen Data

The most accurate multi-layer model has an overall accuracy on unseen data of 90.59%. This is broken up into categories in Figure 5.8. Without Torrenting the average accuracy of the three other categories is 87.40% compared to the previous total of 90.59%.

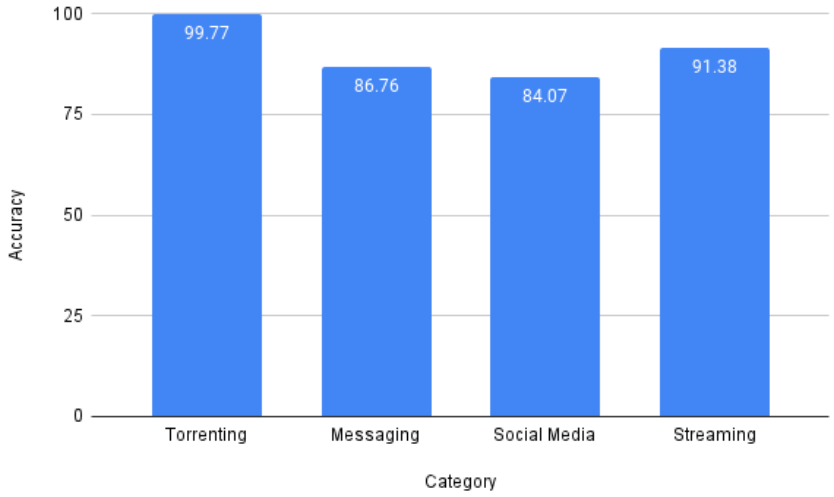

Figure 5.8: Highest performing Multi-layer model's Accuracy per category

5.2.3 Attention-based CNN

As per the results in recorded in Table 5.3, there is no direct correlation between increasing the sampled packets and bytes and the F1 score or unseen data accuracy in the attention-based model category. The best performing model in terms of F1 score and accuracy on unseen data samples 225 bytes from five packets. The confusion matrix for this model classifying unseen data can be seen in Figure 5.9.

Table 5.3: Offline Experimental Results for Attention-based One-dimensional CNNs

Bytes	Packets	Training Accuracy	Validation Accuracy	F1 Score	Unseen Accuracy
144	3	91.66	89.55	88.75	86.94
144	5	93.56	87.91	89.14	85.38
144	7	90.09	89.16	88.22	86.87
196	3	91.06	88.95	88.45	86.76
196	5	94.56	89.61	89.65	87.48
196	7	91.19	89.26	88.45	86.88
225	3	92.89	88.79	89.78	87.30
225	5	95.66	91.04	91.74	89.02
225	7	93.89	89.89	90.53	88.11

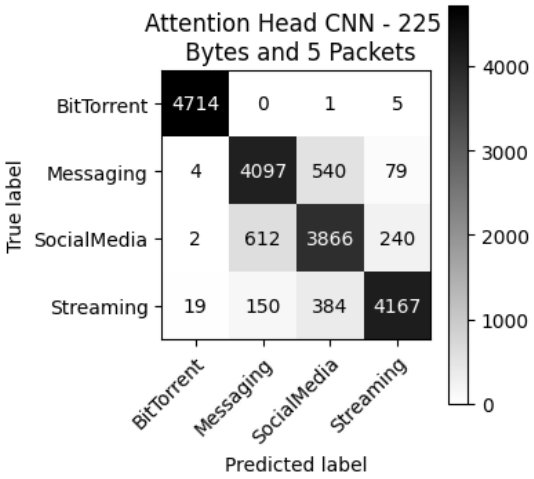

Figure 5.9: Confusion Matrix of Attention-based CNN on Unseen Data

The most accurate attention-based model has a overall accuracy on unseen data of 89.02%. This is broken up into categories in Table 5.10. Without Torrenting the average accuracy of the three other categories is 85.66 percent compared to the 89.02% overall accuracy.

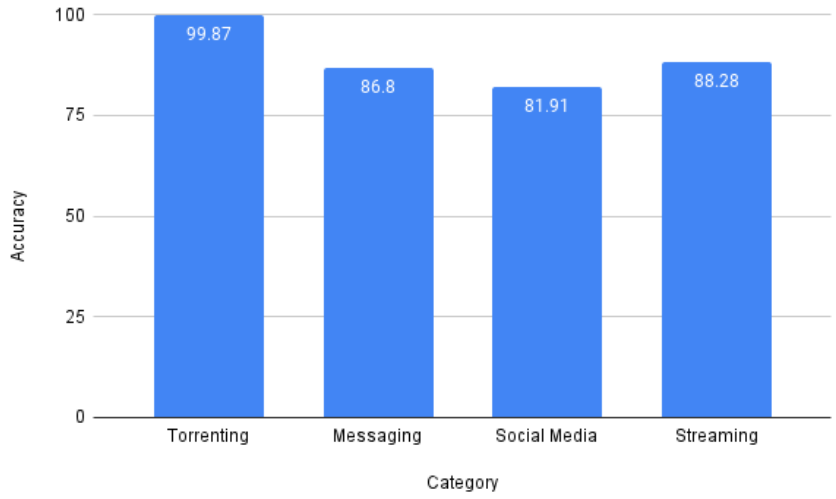

Figure 5.10: Highest Performing Attention-based Model's Accuracy per category

5.2.4 Per Category Model Performance

Figure 5.11 shows the best-performing model from each category. In Figure 5.12, the baseline accuracy of the simplest model, in terms of input, is recorded per model category.

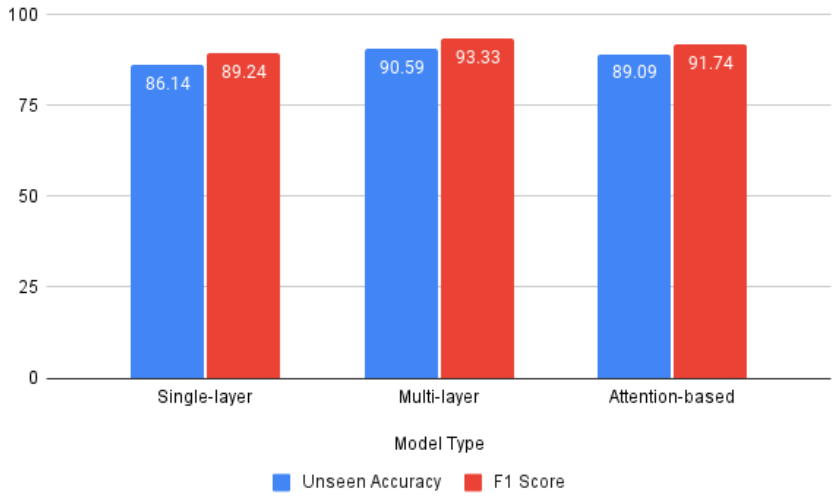

Figure 5.11: Best Performing Model per Category

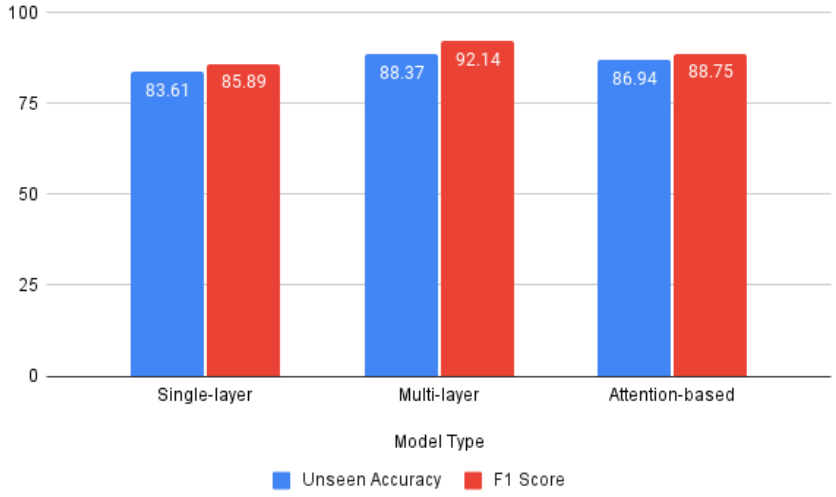

Figure 5.12: Baseline Accuracy per category

When comparing the best performing models in Figure 5.11 and the baseline models in Figure 5.12, it can be seen that the single-layer model's accuracy and F1 score increases by 2.53% and 3.35, respectively. The multi-layer model has an increase in accuracy of 2.22%, and its F1 score increases by 1.19. Finally,

the attention-based model has an increase in accuracy of 2.15% and its F1 score increases by 2.99.

5.3 Online Machine Learning Accuracy

The baseline and best-performing models per category detailed in Section 5.2 are tested in online simulations. Once each model is tested to gather average framework times and model accuracy, the most accurate model, namely the multi-layer model processing 225 bytes and five packets, is used to test the effect a classifier has on network QoS, thus addressing the accuracy and efficiency of machine learning-based classification in real-time.

5.3.1 Online Model-based Results

5.3.1.1 Average Classification Accuracy

The average classification accuracy of the models is not necessarily a fair metric to compare model performance due to the differing number of flows the different network architectures can sustain during the ten-minute simulations. With a differing number of flows, the data sent through the network cannot be balanced, meaning some categories of traffic data will be overrepresented. While differing transformation and classification times will also impact the number of flows replayed. However, to gauge the overall performance of the models individually, accuracy during online tests is an important metric to consider.

The average classification accuracy for each model is depicted in Figure 5.13 across all simulated architectures. The accuracy achieved by each model for each architecture is detailed in Appendix Section A.0.1.

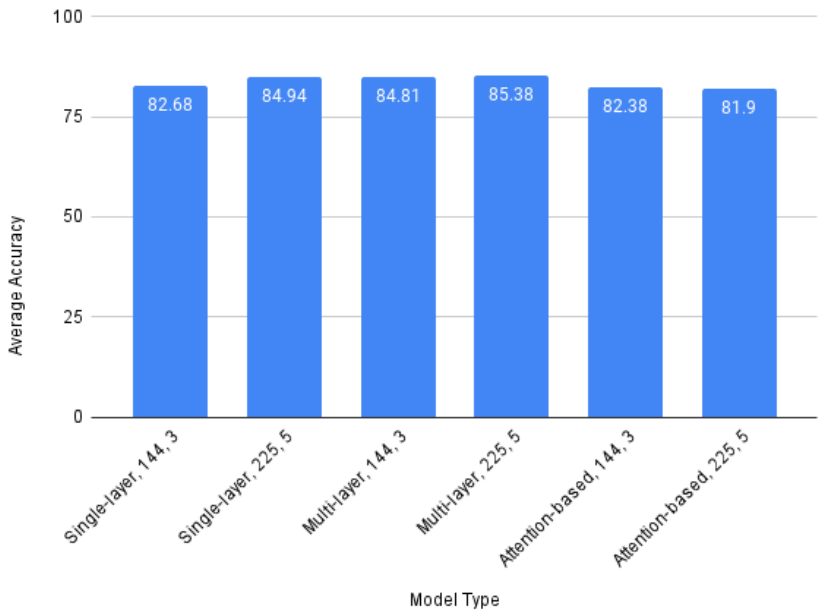

Figure 5.13: Average Online Classification Accuracy

From Figure 5.13, it can be seen that the average accuracy for each model does not fall below 81%, with the attention-based model having the lowest accuracy of 81.90% in real-time classifications. It is also evident that classification accuracy increases with an increase in the number of bytes and packets sampled. This is true in all three categories of models. The highest accuracy of 85.38% is achieved by the Multi-layer model that samples five packets and 225 bytes from each packet. This model is also the most accurate in offline classifications.

5.3.1.2 Average Classification and Transformation Time

Figure 5.14 records the average classification time for each model across all architectures and the average time taken to collect and process x packets. A detailed account of each model's performance per architecture can be found in Appendix Section A.0.1.

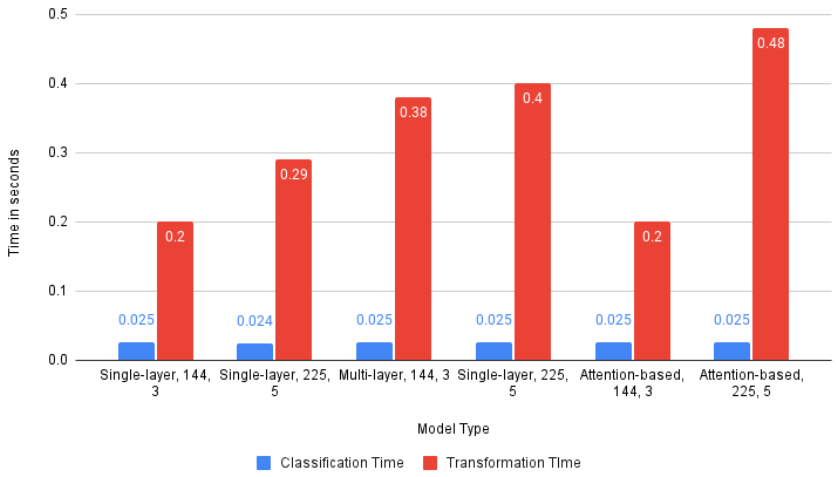

Figure 5.14: Average Online Classification Time and Transformation Time

The average classification time is consistent at 0.025 seconds for five out of the six models. Only one single-layer model, which samples five packets and 225 bytes, has a slightly faster average classification time of 0.024 seconds per flow. This suggests that model complexity does not have a direct impact on classification time in our emulated network. However, when we observe transformation time, it increases across all three model categories with growing input size, as illustrated in Figure 5.14. To break it down: single-layer models show a difference of 45%, multi-layer models have a variance of 5.26%, and attention-based models exhibit a substantial variation of 140%.

The total time from the start of packet collection to a classification can be seen in Figure 5.15. This shows that the simplest CNN, the single-layer model, has the joint lowest total time for a model sampling 144 bytes and three packets and the lowest time for a model sampling five packets and 225 bytes. This can be attributed to the additional CPU resources available due to using a less complex model.

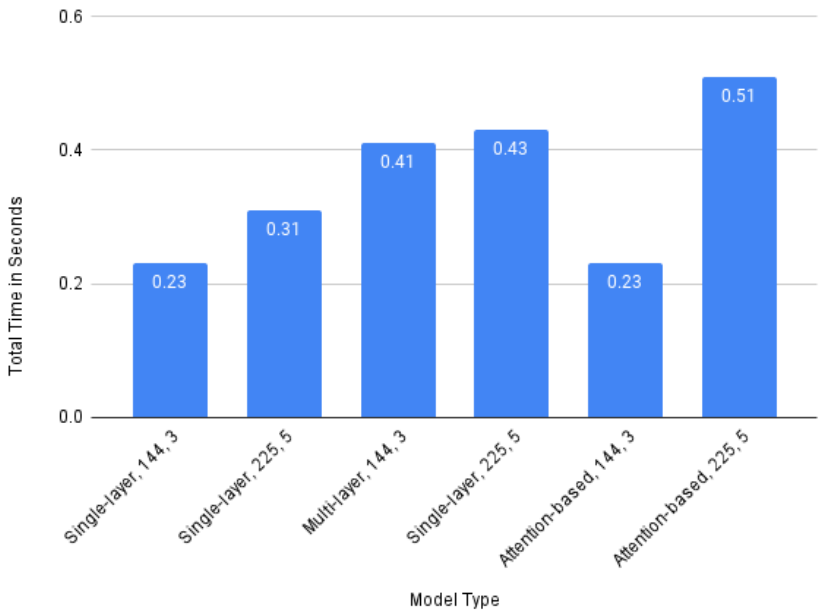

Figure 5.15: Average Time Taken to Gather and Classify Packets

5.3.1.3 Average Flow Adjustment and Meter Entry Time

Figure 5.16 shows the average flow adjustment and meter entry time for each model tested. The average flow adjustment time is the time taken to make an API call to the ONOS controller and receive a successful notification that the requested flow adjustments have been made. The average meter entry time is the time to add meter entries to each switch before network simulations begin. A detailed breakdown of these metrics can be found for each architecture and model in Appendix Section A.0.1.

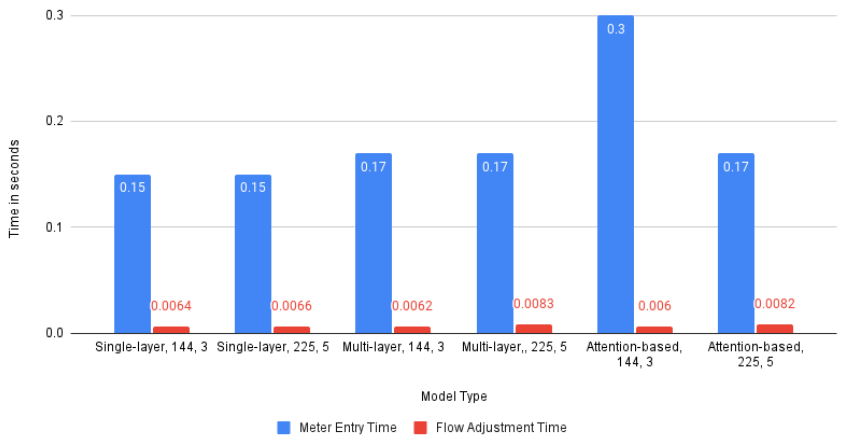

Figure 5.16: Average Online Flow Adjustment and Meter Entry Time

From Figure 5.16, it is clear that there is a direct correlation between the increase in packets and bytes sampled and the flow adjustment time. There is an increase in adjustment time of 3.13%, 33.87% and 36.67% between the models in the single-layer, multi-layer and attention-based architectures, respectively. This can be attributed to the increase in CPU resources required by the classifier.

There is no relationship between meter entry time and model input complexity, as expected, as it happens before classification starts, with the single-layer and multi-layer architectures having the same averages for both models tested. However, there is a significant difference in meter entry time between the two attention-based models. An increase in model complexity coincided with a 43.33% decrease in average meter entry time. While this value is not indicative of any models' performance, it is a necessary time constraint placed on the SDN to allow classification.

5.3.2 Average Framework Time

The average time taken from the first packet of a flow being received by the classifier to a successful flow adjustment based on the classifier's classification is

detailed in Figure 5.17.

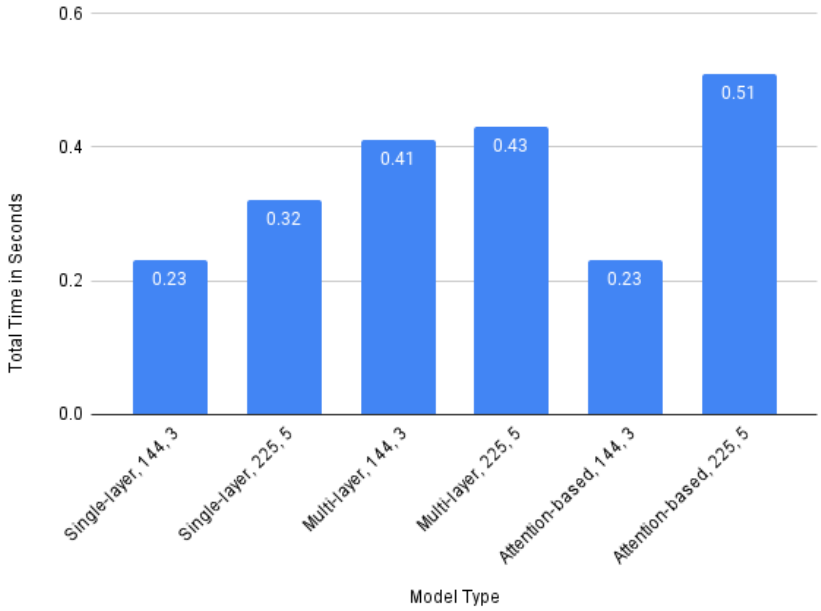

Figure 5.17: Average Total Framework Time

Based on the results depicted in Figure 5.17, there is a correlation between an increase in input complexity and the flow adjustment time with every category increasing as the number of packets and bytes sampled increases. The increases are 39.13%, 4.88% and 121.74% for the single-layer, multi-layer and attention-based models.

5.4 Quality of Service Results

The QoS of SDN networks can be measured with monitoring and control mechanisms built into the OpenFlow protocol and SDN controllers. Flow tables and meters allow per-flow rate-limiting that can be applied dynamically to traffic based on the classifier's classifications. This rate-limiting will directly affect the packet loss and flow throughput throughout the network. These are the QoS

metrics that are recorded during the online QoS testing and allow the viability of the overall framework to be analysed.

5.4.1 Average Network Throughput

Network throughput can be seen in Figure 5.18 with a detailed breakdown of the per category throughput in Table 5.4.

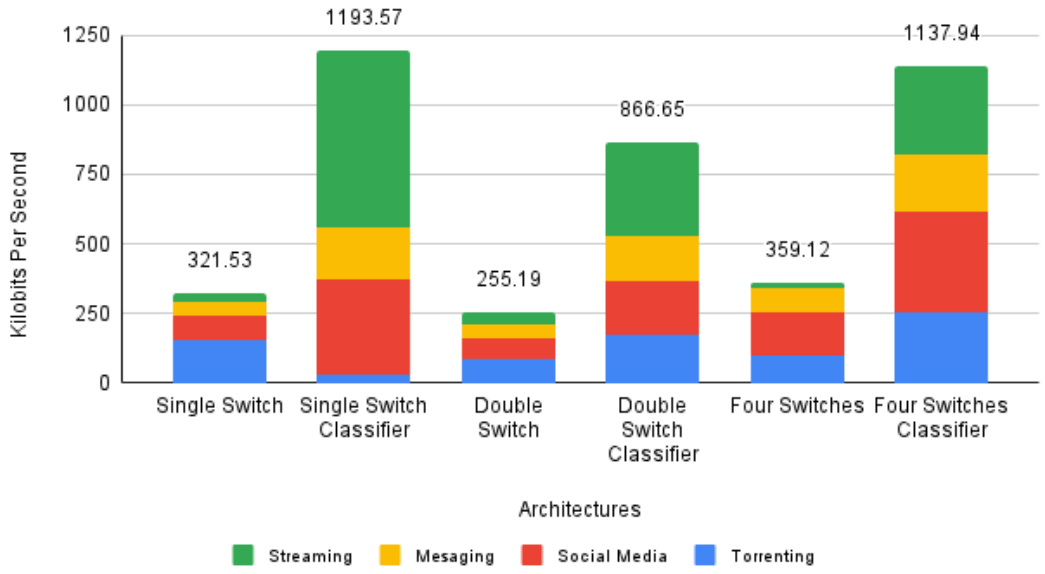

Figure 5.18: Average Network Throughput per Flow per Architecture

It can be seen from Figure 5.18 that the average flow throughput of each architecture increases by more than 200% when adding a classifier to the network. The largest increase is in the single switch architecture, where the overall throughput increases by 271.22%, followed by an increase of 239.61% in the double switch architecture. Finally, the smallest increase is 216.87% in the four-switch architecture. Flow throughput is limited by the link capacity of 2000 kbps between each switch and the network's backhaul speed of 2000 kbps without a classifier and limited by the rate limits applied to each flow by the classifier in the classifier-enabled

networks. From Figure 5.18, it can be seen that adding a classifier increases the average flow throughput utilisation of link capacity from 16.08% to 59.83% in the single switch architecture, from 12.76% to 43.33% in the double switch architecture and from 17.96% to 56.90% in the four switch architecture. This represents a 43.75%, 30.57% and 38.94% increase in average flow throughput utilisation for the single-switch, double-switch and four-switch architecture, respectively. The differences in average flow throughput between architectures are due to the ratio of clients to switches changing, and thus the link capacity available to each client changing. This results in differing resource availability and different proportions of traffic categories replayed in each simulation.

Table 5.4: Network Throughput per Flow per Architecture

Architecture	Messaging	Streaming	Social Media	Torrenting	Total
Single Switch	51.17	30.33	87.99	152.04	321.53
Single Switch Classifier	189.06	631.23	344.21	29.07	1193.57
Double Switch	46.83	45.98	74.96	87.42	255.19
Double Switch Classifier	157.48	341.40	194.25	173.52	866.65
Four Switches	84.90	19.94	156.7	97.58	359.12
Four Switches Classifier	204.9	316.06	364.74	252.24	1137.94

Table 5.4 shows that average throughput increases in every traffic category when using a classifier except in the case of the Torrenting category in the single switch architecture, where there is a decrease of 80.88%. When analysing these increases, the largest increase in throughput is in the streaming category in each

network architecture. The Streaming category increases by 1981.23%, 642.50% and 1485.06% in the single, double and four switch architectures, respectively. The streaming category also records the highest average throughput per category in two of the three classifier-based architectures. In the single switch architecture, its total of 631.23 kbps is 287.02 kbps higher than the Social Media category, which is in second place. While in the double switch architecture, the total of 341.40 kbps is 147.15 kbps higher than the Social Media category, which is the second highest category in average throughput. The highest-performing category in the four-switch architecture is Social Media, with a throughput of 364.74 kbps, compared to Streaming in second place with a throughput of 316.06 kbps.

5.4.2 Average Network Packet Loss

While network packet loss would be an issue if it is unexpected, in the case of the experimental networks with classifiers, there is purposeful packet rate limiting which drops packets to keep the flow throughput below a specified rate. Additionally, the packet loss is expected to be high due to the use of Tcpreplay in the experimental setup, detailed in Section 4.4.2. The average packet loss per switch can be seen in Figures 5.19, 5.20 and 5.21.

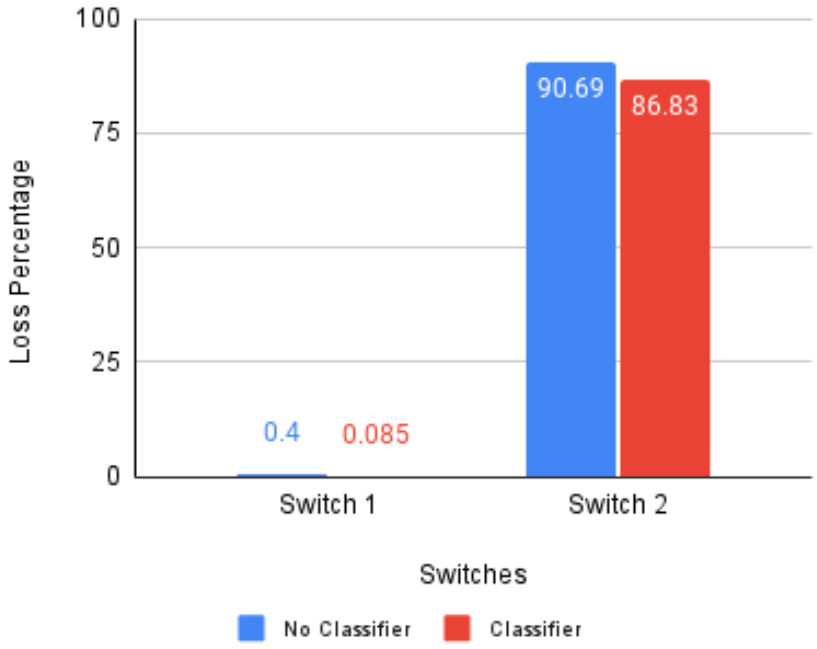

Figure 5.19: Average Packet Loss for the Single Switch Architecture

Figure 5.19 depicts the packet loss for the single switch architecture. Focusing on the gateway switch, switch one, there is a 0.315% decrease in packet loss from 0.4% without a classifier to 0.085% with a classifier.

Switch 2 is connected to clients and has high packet loss percentages in both architectures due to the link capacity in both architectures and the rate-limiting applied in the classifier-enabled architecture. Switch 2 has a packet loss percentage of 90.69% without a classifier and 86.83% with a classifier.

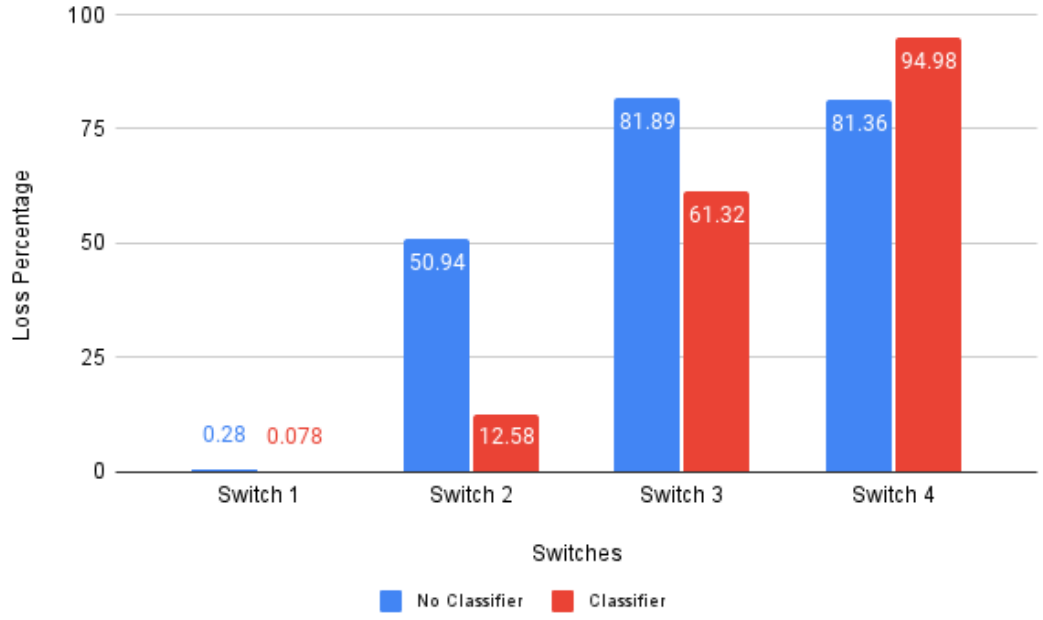

Figure 5.20: Average Packet Loss for the Double Switch Architecture

In Figure 5.20, the packet loss for the double switch architecture is plotted. Switch one is the gateway switch, and switch two is an internal switch not connected to clients. These switches show a packet loss reduction when a classifier is added to the architecture. The gateway switch's packet loss decreases by 0.202% from 0.28% to 0.078%. While switch two decreases by 38.36% from 50.94% to 12.58%. Switch 3 and 4 are connected to clients and have high packet loss percentages in both architectures due to the link capacity in both architectures and the rate-limiting applied in the classifier-enabled architecture. Switch 3 and 4 have a packet loss percentage of 81.89% and 81.36% without a classifier and 61.32% and 94.98% with a classifier.

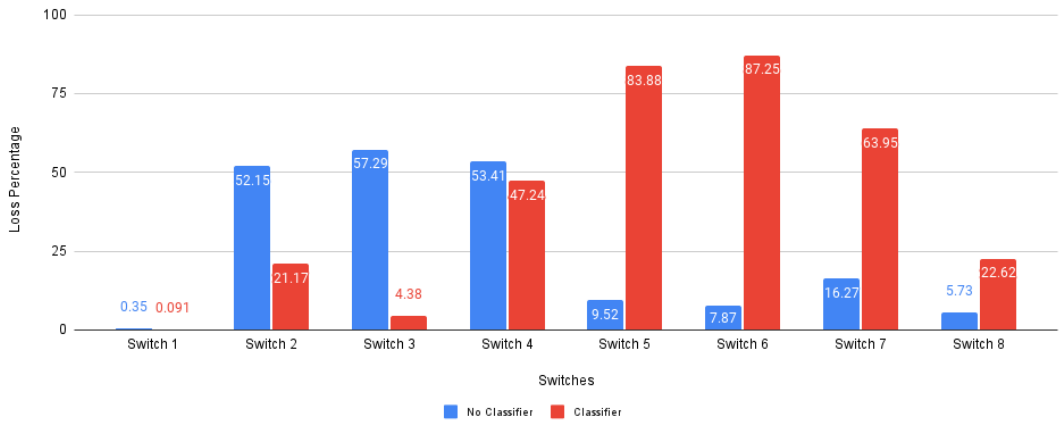

Figure 5.21: Average Packet Loss for the Four Switch Architecture

Figure 5.21 details the packet loss per switch in the four-switch architecture. Switch one is the gateway switch, and switches two, three and four are internal switches not connected to clients. All four switches experience a decrease in packet loss when a classifier is added to the architecture. At the gateway switch, packet loss decreases by 0.259% from 0.35% to 0.091% while switches 2, 3 and 4 decrease by 30.98%, 52.91% and 6.17%, respectively.

Switch 5, 6, 7 and 8 are connected to clients and have lower packet loss percentages in both architectures than the previously examined client-facing switches. This is because, on average, there are fewer clients at each switch and, thus, less inbound traffic. Switch 5, 6, 7 and 8 have a packet loss percentage of 9.52%, 7.78%, 16.27% and 5.76% without a classifier and 83.88%, 87.25%, 63.95% and 22.62% with a classifier. While packet loss decreases at internal switches when adding a classifier, it is still high in all architectures. Packet loss is high due to the incoming links to each port on the switch having a capacity of 2 Mbps and the single outwards facing port also having a capacity of 2 Mbps. This, in conjunction with a fixed traffic replay speed of 2 Mbps at each client, causes the outwards facing ports to become bottlenecks, resulting in the high packet loss

visible in Figures 5.19, 5.20 and 5.21. This packet loss would be significantly lower in a real-world network as the transport layer would reduce the sending rate when the switches enforce link rate limiting.

5.5 Summary

From the analysis of the iNethi dataset, it is evident that YouTube flows are the largest, in terms of megabits, across both iNethi datasets. Facebook is the most frequently occurring application in the training dataset, and BitTorrent is the most frequently occurring application in the testing dataset.

The above sections show that the most accurate models have an input of 225 bytes and five packets across all model categories. Torrenting is the most accurately classified traffic category in these top-performing models, followed by Streaming. The classification accuracy of the best-performing models in each category displayed in Table 5.5 shows that each model exceeded the 85.8% accuracy achieved by Wang et al. [7] in related work with an accuracy of 86.14%, 90.59% and 89.09% for the single-layer, multi-layer and attention-based models respectively.

Table 5.5: Highest Offline Experimental Accuracy on Unseen Data Per Model Type

Model Type	Percentage Accuracy
Single-layer	86.14
Multi-layer	90.59
Attention-based	89.0

The QoS-related results show an increase in network bandwidth utilisation when a classifier is added to the architecture and a reduction in packet loss at internal switches. The packet loss values are higher than what would occur in a real-world network due to the limitations of the testing environment.

Chapter 6

Discussion and Conclusions

This section breaks down the performance of the machine learning models and ties this into the QoS metrics detailed in Chapter 5. Conclusions on the viability of the proposed QoS framework are made, and thus the proposed research questions are answered.

6.1 Model Accuracy and Classification Time

During the training and testing of the machine learning models, 27 models are examined. These models are split into three categories, single-layer, multi-layer and attention-based one-dimensional CNNs. This phase of the study partly answers the first research question: "Are lightweight deep learning models feasible for real-time traffic classification in a resource-constrained SDN?". Accurate offline results prove that these models can classify encrypted network traffic, while their performance in real-time is examined in the online tests.

When comparing the three models in terms of classification accuracy, it can be seen that more complex models, i.e. the attention-based and multi-layer models, outperform the basic single-layer model. Thus, adding complexity increases

accuracy. Additionally, the best-performing models from each category use the same input structure of 225 bytes and five packets and outperform their corresponding base models using 144 bytes and three packets. However, using 225 bytes and seven packets did not yield better accuracy. Thus, increasing the number of bytes and packets sampled correctly results in better accuracy. Therefore, it can be seen that increasing model and input complexity has a strong positive correlation with model accuracy.

Zou et al. [8] hypothesize that incorporating a multi-head attention layer will allow for the time-based relationships between packets to be analysed with links between packets in any positions created due to the projected spaces these models work in. However, this category's most accurate model does not outperform the multi-layer architecture's most accurate model in F1 score or unseen accuracy. This indicates that the time-sequence dependencies that can be extended beyond the sequential packets using an attention-head module are not as significant as Zou et al. [8] hypothesised when working in the input space this study focuses on. The multi-layer model is 1.59% more accurate on unseen data and has an F1 score 1.5 points higher than the attention-based model.

When analysing the confusion matrices for the best model per architecture presented in Section 5.2, the results are as shown in Table 6.1.

Table 6.1: Classification Accuracy per Category

Architecture	Messaging	Streaming	Social Media	Torrenting
Single-layer	76.45	86.19	82.75	99.56
Multi-layer	86.76	91.38	84.07	99.77
Attention-based	86.80	88.28	81.91	99.87

Across all models, Torrenting is the most accurately classified class of traffic, followed by Streaming. The Torrenting category is only made up of Bittorrent traffic and is thus an application-specific category resulting in very high accu-

racy compared to the other categories. Social Media and Messaging are most frequently mislabeled as each other across all three models, with 20.89%, 12.12% and 11.44% of Social Media traffic being classified as Messaging content by the single-layer, multi-layer and attention-based models, respectively. On the other hand, 11.50%, 12.61% and 12.97% of Messaging traffic are classified as Social Media traffic by the single-layer, multi-layer and attention-based models. This can be assumed to be caused by the large amount of text-based content that will make up this traffic and the fact that all traffic that makes up these categories is generated by applications created and hosted by Meta[1].

The first research question can be definitively answered when moving to online tests using the best-performing models found in offline tests. While the hypothesised accuracy of attention-based models does not result in the most accurate model, they also offer efficiency as the projected spaces they work in can be processed in parallel. However, this made no noticeable difference in real-time traffic classification. Each model tested had a classification time of 0.025 seconds except for the single-layer model processing 225 bytes and five packets which is 0.001 seconds faster with an average classification time of 0.024 seconds across the three architectures. Therefore, the hypothesised benefits of adding a multi-head attention module do not result in a faster or more accurate classifier.

The input that a model processes has a direct correlation with the time taken for a prediction to be made. While the aforementioned classification times are stable, there is an increase in the time taken to gather and transform packets into a suitable format as the number of packets and bytes increases. Across all three model types, there is an increase in transformation and gathering time as the number of packets and bytes increases. On average, the slowest model can gather and classify a flow every 0.51 seconds. This results in overall framework

[1]https://about.meta.com/

times that can enforce QoS reservations in under a second.

6.2 Quality of Service Results

Throughput and packet loss are the two QoS metrics that are gathered and affected by adding a classifier to the network architecture. Online tests are conducted to assess these metrics with and without a classifier. The second and third research questions can be answered by analysing these results. Where the second research question is framed as "What overhead is incurred, in terms of network QoS, when adding a real-time traffic classifier to a resource-constrained SDN?" and the third is framed as "Can existing, pre-packaged SDN QoS management tools use real-time traffic classification data to improve network QoS?".

The online tests show that the average throughput per second significantly increases when adding a classifier to all test architectures. The online tests also show packet loss at non-rate-limited switches decreases when adding a traffic classifier. Thus, adding a classifier in these test architectures increases network QoS. This positive effect on QoS is caused by the improved resource allocation and flow management offered by a classifier-enabled architecture. This is evident in the Streaming category, which is prioritised and allocated the most resources in the experiments. The average flow throughput for the streaming category increases by 1981.23%, 642.50% and 1485.06% across the single, double and four switch architecture. Packet loss decreases at non-rate-limited switches due to a decrease in bandwidth usage as all inbound traffic to these switches originates from a switch where rate-limiting has taken place. This decrease in bandwidth usage means more link capacity is available, resulting in fewer dropped packets at internal switches. In terms of throughput, a decrease in packet loss and an increase in the availability of link capacity means more traffic can pass from a

host to the gateway switch successfully.

It is important to note that high packet loss values are expected as all hosts are programmed to replay traffic at a fixed speed of 2 Mbps to keep the replay speed a controlled variable. The Mininet link capacity is also set to 2 Mbps to ensure link capacity is a controlled variable. The packet loss is caused by Scapy, the network traffic analysis and replay library used in this study, which requires a rate in Packets Per Second (PPS) or Mbps if the original timestamps of the PCAP file are not to be used for traffic replay. Therefore, if a packet is dropped, it is not retransmitted in the case of TCP traffic, and the traffic rate is not stalled. Instead, the next packet is replayed.

It is also vital to compare the theoretical rate limits applied to each category of traffic and the average throughput achieved across the online experiments. These are depicted in Figure 6.1. From Figure 6.1, it is apparent that three of the four traffic categories do not get close to their theoretical limit. Messaging is the only category within ten percentage points of its theoretical limit. Streaming achieves an average throughput of 429.56 kbps with a theoretical limit of 1000 kbps. While this category has the most resources reserved for it and achieves the highest throughput, the theoretical limit is 570.44 kbps higher than the achieved rate. This means that, on average, less than 50% of the allocated resources to this category are used. Additionally, both the Social Media and Torrenting categories use more resources than are allocated to them. Torrenting exceeds its allocation by 51.61 kbps, and Social Media exceeds its allocation by 101.07 kbps. The Torrenting category has the lowest throughput and is allocated the least resources, which is significant in terms of QoS.

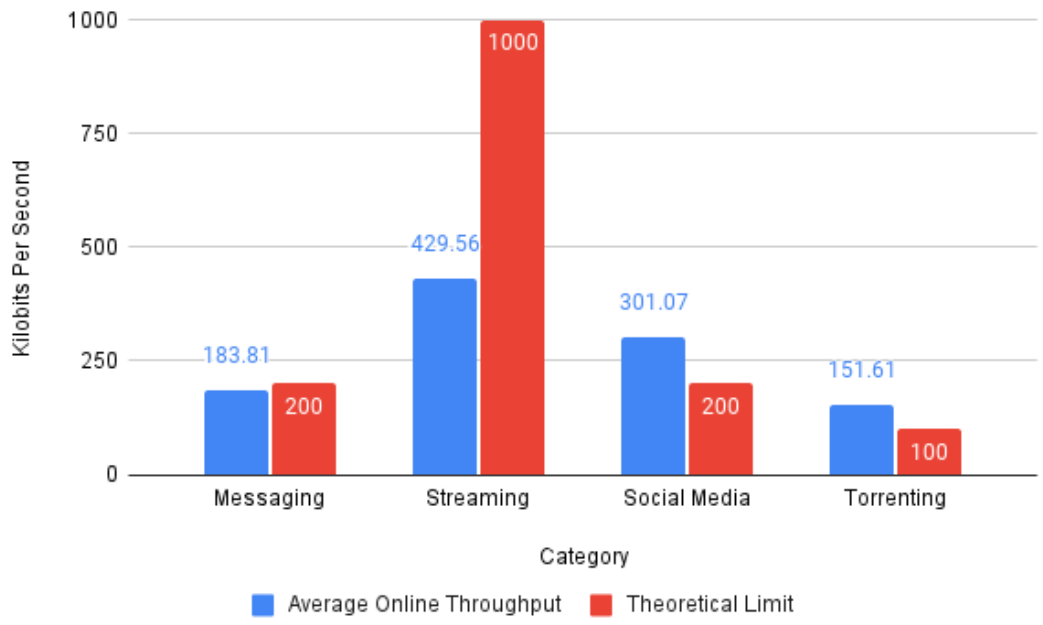

Figure 6.1: Theoretical Limit versus Achieved Online Throughput

This difference in resource allocation can be attributed to the average flow size in the online test dataset. As per the results in Section 5.1.2, the average flow sizes are summarised and displayed in Table 6.2. While the machine learning model used in online tests is accurate, it takes an average of 0.43 seconds for the framework to classify and create a flow entry. This means that the longer the flow, the greater the effect of the rate-limiting. Thus, the small flows relative to the replay speed of 2 Mbps would result in large portions of the flow being sent before rate-limiting would be applied and thus result in high packet loss at the internal switches as the packets would be sent from every host at a constant rate of 2 Mbps. This means that the full capabilities of this framework are not being taken advantage of in the online tests as most of the traffic flows are short-lived, and thus resource allocations cannot be applied to them. The significant increase in average flow throughput for the Streaming category when adding a

classifier can be attributed to the fact that it has the highest average flow size of 2.9 Mbps, 262.5% larger than the second-highest average flow size of 0.8 Mbps for the Torrenting category.

Category	Average Flow Size (Mbps)
Messaging	0.023
Streaming	2.9
Social Media	0.54
Torrenting	0.8

Table 6.2: Average Flow size per category in the Online Test Dataset

6.3 Viability and Effectiveness of Real-time Traffic Classification

The low overhead added to the network in the experiments when adding a classifier and the high classification accuracy of lightweight deep-learning models verify that using these models for real-time traffic classification is viable. Therefore, these models can be used as the foundation of a real-time traffic classification framework that adds negligible overhead while increasing network QoS. Network overhead is added when duplicating traffic and forwarding it to the classifier. Added resources are needed to run the classifier. However, this does not negatively affect the network's QoS across all test architectures. Given that these models are lightweight and run effectively using only a CPU, they are suitable for resource-constrained environments where the added QoS benefits are crucial to providing adequate QoS. However, a GPU is needed for model training. This means these models are effectively out of reach in low-resource networks unless outside resources are used to train the machine-learning models.

The APIs and tools needed to enable dynamic resource allocation and bandwidth reservation are built-in to the existing SDN protocol, OpenFlow, meaning

no overhead is added in this regard. Thus, the addition of the framework designed in this study will allow network administrators to improve network QoS with negligible overhead once the network is SDN-enabled.

However, the QoS results show that networks that share traffic patterns with the iNethi CWN, i.e. short-lived flows, may not be able to take full advantage of the capabilities of this particular framework.

6.4 Conclusions and Future Work

In a CWN, where Internet access bridges the digital divide and provides access to essential resources, improving the users' experience and enabling stable connection to the Internet is invaluable. In under-resourced areas, like Ocean View, adding a framework that allows network administrators to limit resource-heavy categories such as torrenting and provide equitable access to the community is pivotal in solidifying the use of CWNs in communities. An improved user experience will result in community buy-in and enable growth and sustainability.

For this framework to become a reality and be deployed in real-world networks, future work needs to be done. Testing the framework with longer flows will allow for the framework's impact on QoS to be examined more extensively. Additionally, testing the framework in a real-world network environment with real traffic would eliminate the packet loss caused by the static nature of Scapy's traffic replay function.

Appendix A

Experimental Results

A.0.1 Online Results

A.0.1.1 Single-layer Classification Accuracy

Table A.1: Online Classification Accuracy for Single-layer three packet, 144 byte model

Switches	Hosts	Unseen Accuracy
1	15	88.31
2	15	79.78
4	15	79.95

Table A.2: Online Classification Accuracy for Single-layer five packet, 225 byte model

Switches	Hosts	Unseen Accuracy
1	15	89.66
2	15	83.57
4	15	81.59

A.0.1.2 Multi-layer Classification Accuracy

Table A.3: Online Classification Accuracy for Multi-layer three packet, 144 byte model

Switches	Hosts	Unseen Accuracy
1	15	86.89
2	15	85.58
4	15	81.96

Table A.4: Online Classification Accuracy for Multi-layer five packet, 225 byte model

Switches	Hosts	Unseen Accuracy
1	15	95.35
2	15	77.25
4	15	83.54

A.0.1.3 Attention-based Classification Accuracy

Table A.5: Online Classification Accuracy for Attention-based three packet, 144 byte model

Switches	Hosts	Unseen Accuracy
1	15	87.01
2	15	79.45
4	15	80.68

Table A.6: Online Classification Accuracy for Attention-based five packet, 225 byte model

Switches	Hosts	Unseen Accuracy
1	15	87.23
2	15	78.66
4	15	79.52

A.0.1.4 Single-Layer Time-based Metrics

Table A.7: Time-based Metrics for Single-layer Three packet, 144 byte model

Switches	Hosts	Classification Time	Flow Adjustment Time	Transformation Time
1	15	0.025	0.006	0.24
2	15	0.025	0.007	0.20
4	15	0.025	0.0062	0.16

Table A.8: Time-based Metrics for Single-layer Five packet, 225 byte model

Switches	Hosts	Classification Time	Flow Adjustment Time	Transformation Time
1	15	0.025	0.0068	0.34
2	15	0.025	0.006	0.32
4	15	0.024	0.007	0.21

A.0.1.5 Multi-Layer Time-based Metrics

Table A.9: Time-based Metrics for Multi-layer Three packet, 144 byte model

Switches	Hosts	Classification Time	Flow Adjustment Time	Transformation Time
1	15	0.025	0.0063	0.52
2	15	0.025	0.0062	0.40
4	15	0.025	0.0061	0.22

Table A.10: Time-based Metrics for Multi-layer Five packet, 225 byte model

Switches	Hosts	Classification Time	Flow Adjustment Time	Transformation Time
1	15	0.025	0.0086	0.49
2	15	0.024	0.0083	0.4
4	15	0.025	0.008	0.31

A.0.1.6 Attention-based Time-based Metrics

Table A.11: Time-based Metrics for Attention-based Three packet, 144 byte model

Switches	Hosts	Classification Time	Flow Adjustment Time	Transformation Time
1	15	0.025	0.0061	0.22
2	15	0.025	0.006	0.18
4	15	0.025	0.0059	0.20

Table A.12: Time-based Metrics for Attention-based Five packet, 225 byte model

Switches	Hosts	Classification Time	Flow Adjustment Time	Transformation Time
1	15	0.024	0.008	0.72
2	15	0.025	0.0086	0.42
4	15	0.025	0.008	0.30

References

[1] B. Braem, C. Blondia, C. Barz, H. Rogge, F. Freitag, L. Navarro, J. Bonicioli, S. Papathanasiou, P. Escrich, R. Baig Viñas, A. L. Kaplan, A. Neumann, I. Vilata i Balaguer, B. Tatum, and M. Matson, "A case for research with and on community networks," *SIGCOMM Comput. Commun. Rev.*, vol. 43, p. 68–73, jul 2013. 1

[2] A. Neumann, E. López, and L. Navarro, "Evaluation of mesh routing protocols for wireless community networks," *Computer Networks*, vol. 93, pp. 308–323, 2015. Community Networks. 1

[3] K. Kirkpatrick, "Software-defined networking," *Communications of the ACM*, vol. 56, no. 9, pp. 16–19, 2013. 1

[4] M. Karakus and A. Durresi, "Quality of service (qos) in software defined networking (sdn): A survey," *Journal of Network and Computer Applications*, vol. 80, pp. 200–218, 2017. 1, 2, 15, 16, 20

[5] A. Malik, R. de Fréin, M. Al-Zeyadi, and J. Andreu-Perez, "Intelligent sdn traffic classification using deep learning: Deep-sdn," in *2020 2nd International Conference on Computer Communication and the Internet (ICCCI)*, pp. 184–189, 2020. 1, 2, 12

[6] J. Cheng, R. He, E. Yuepeng, Y. Wu, J. You, and T. Li, "Real-time encrypted

traffic classification via lightweight neural networks," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2020. 2, 8, 12, 13, 22, 23, 30, 31

[7] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *2017 IEEE international conference on intelligence and security informatics (ISI)*, pp. 43–48, IEEE, 2017. 2, 8, 12, 22, 23, 30, 31, 65

[8] Z. Zou, J. Ge, H. Zheng, Y. Wu, C. Han, and Z. Yao, "Encrypted traffic classification with a convolutional long short-term memory neural network," in *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 329–334, IEEE, 2018. 2, 8, 11, 12, 22, 23, 30, 31, 33, 67

[9] M. R. Lorini, M. Densmore, D. Johnson, S. Hadzic, H. Mthoko, G. Manuel, M. Waries, and A. v. Zyl, "Localize-it: Co-designing a community-owned platform," in *International Development Informatics Association Conference*, pp. 243–257, Springer, 2018. 5, 6, 7

[10] A. van Zyl and D. L. Johnson, "Inethi: Locked down but not locked out," *XRDS*, vol. 27, p. 54–57, dec 2020. 5, 6, 7, 37

[11] B. Braem, C. Blondia, C. Barz, H. Rogge, F. Freitag, L. Navarro, J. Bonicioli, S. Papathanasiou, P. Escrich, R. Baig Vinas, *et al.*, "A case for research with and on community networks," 2013. 5, 6

[12] P. A. Frangoudis, G. C. Polyzos, and V. P. Kemerlis, "Wireless community

networks: an alternative approach for nomadic broadband network access," *IEEE Communications Magazine*, vol. 49, no. 5, pp. 206–213, 2011. 6

[13] A. Phokeer, S. Hadzic, E. Nitschke, A. Van Zyl, D. Johnson, M. Densmore, and J. Chavula, "inethi community network: A first look at local and internet traffic usage," in *Proceedings of the 3rd ACM SIGCAS Conference on Computing and Sustainable Societies*, pp. 342–344, 2020. 6

[14] A. R. Mohammed, S. A. Mohammed, and S. Shirmohammadi, "Machine learning and deep learning based traffic classification and prediction in software defined networking," in *2019 IEEE International Symposium on Measurements & Networking (M&N)*, pp. 1–6, IEEE, 2019. 8, 9, 10, 11, 13, 14

[15] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning - Data Mining, Inference, and Prediction.* Stanford, California, USA: Springer, 2009. 8

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. 9

[17] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, *et al.*, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, 2018. 9

[18] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng, "End-to-end text recognition with convolutional neural networks," in *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*, pp. 3304–3308, IEEE, 2012. 9

[19] S. Haider, A. Akhunzada, I. Mustafa, T. B. Patel, A. Fernandez, K.-K. R. Choo, and J. Iqbal, "A deep cnn ensemble framework for efficient ddos attack detection in software defined networks," *Ieee Access*, vol. 8, pp. 53972–53983, 2020. 10

[20] L. Medsker and L. C. Jain, *Recurrent neural networks: design and applications*. CRC press, 1999. 10

[21] I. Sutskever, *Training recurrent neural networks*. University of Toronto, Toronto, Canada, 2013. 11

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. 11

[23] C. Gutterman, K. Guo, S. Arora, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman, "Requet: Real-time qoe detection for encrypted youtube traffic," in *Proceedings of the 10th ACM Multimedia Systems Conference*, pp. 48–59, 2019. 12

[24] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features.," in *ICISSp*, pp. 253–262, 2017. 13

[25] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 393–430, 2018. 14, 15

[26] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in sdn home gateway," *IEEE Access*, vol. 6, pp. 55380–55391, 2018. 14

[27] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014. 14

[28] J. McCauley, A. Panda, M. Casado, T. Koponen, and S. Shenker, "Extending sdn to large-scale networks," *Open Networking Summit*, pp. 1–2, 2013. 14

[29] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "Sdn controllers: A comparative study," in *2016 18th mediterranean electrotechnical conference (MELECON)*, pp. 1–6, IEEE, 2016. 15

[30] L. Zhu, M. M. Karim, K. Sharif, C. Xu, F. Li, X. Du, and M. Guizani, "Sdn controllers: A comprehensive analysis and performance evaluation study," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–40, 2020. 15

[31] A. Binsahaq, T. R. Sheltami, and K. Salah, "A survey on autonomic provisioning and management of qos in sdn networks," *IEEE Access*, vol. 7, pp. 73384–73435, 2019. 16, 17, 19, 20, 41

[32] O. N. Foundation, "Openflow switch specification: Version 1.3.0," 2012. 16, 17, 20

[33] J. Yan and D. Jin, "Vt-mininet: Virtual-time-enabled mininet for scalable and accurate software-define network emulation," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, pp. 1–7, 2015. 17, 18

[34] M. P. Contributors, "Mininet overview," jul 2021. 18

[35] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto, "Identifying elephant flows through periodically sampled packets," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pp. 115–120, 2004. 27

[36] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–8, IEEE, 2014. 39, 40, 41

[37] Q. He and S. Wang, "A low-cost measurement framework in software defined networks," *International Journal of Communications, Network and System Sciences*, vol. 10, no. 5, pp. 54–66, 2017. 39, 41

[38] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pp. 407–414, sn, 2016. 22

[39] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pp. 407–414, 2016. 22

[40] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula, "Automated and scalable qos control for network convergence.," *INM/WREN*, vol. 10, no. 1, pp. 1–1, 2010. 21

[41] I. Bueno, J. I. Aznar, E. Escalona, J. F. Riera, and J. A. G. Espín, "An opennaas based sdn framework for dynamic qos control," *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, pp. 1–7, 2013. 21

[42] Q. Duan, "Network-as-a-service in software-defined networks for end-to-end qos provisioning," *2014 23rd Wireless and Optical Communication Conference (WOCC)*, pp. 1–5, 2014. 21

[43] M. S. Seddiki, M. Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster, and Y.-Q. Song, "Flowqos: Per-flow quality of service for broadband access networks," tech. rep., Georgia Institute of Technology, 2015. 21

[44] M. S. Seddiki, M. Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster, and Y.-Q. Song, "Flowqos: Qos for the rest of us," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 207–208, 2014. 21

[45] R. B. Santos, T. R. Ribeiro, and C. A. C. César, "A network monitor and controller using only openflow," *2015 Latin American Network Operations and Management Symposium (LANOMS)*, pp. 9–16, 2015. 20

[46] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "Opentm: traffic matrix estimator for openflow networks," in *International Conference on Passive and Active Network Measurement*, pp. 201–210, Springer, 2010. 19, 20

[47] B. Siniarski, C. Olariu, P. Perry, and J. Murphy, "Openflow based voip qoe monitoring in enterprise sdn," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 660–663, 2017. 20