Real-Time Application of Deep Learning to Intrusion Detection in 5G-Multi-Access Edge Computing

Omesh Anthony Fernando

School of Engineering and Computer Science
University of Hertfordshire

Submitted to the University of Hertfordshire in partial fulfilment of the requirement of the degree of Doctor of Philosophy

May 2023

I would like to dedicate this thesis to Jesus Christ, my Lord and Saviour ...
"How Great Thou Art" (Boberg, Carl 1885) ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 60,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Omesh Anthony Fernando
May 2023

Acknowledgements

I would like to thank the God Almighty for the continuous guidance, blessings and love showered upon me. Next, I would like to convey my gratitude to the University of Hertfordshire for providing me with a studentship to achieve a PhD in Computer Science.

I would like to convey my sincerest appreciation to my supervisors: Dr Hannan Xiao, Dr William Joseph Spring and Dr Xianhui Che. The wealth of knowledge, experience, motivation, support and guidance that they have given me is immeasurable.

I would like to convey my gratitude to my late father, Mr Royce Fernando. To my loving, beautiful, and kind mother Mrs Emarley Fernando and Dr Mesaj Fernando my brother, whose guidance has always been the biggest pillar of strength. I also thank my colleagues in STRI, for all the times we shared our struggles, laughter and jokes during this journey of PhD. Also, I thank all the students I have come across, who helped enrich my knowledge. Lastly, to my betrothed, Jayanie, the irreplaceable treasure of my life, for her unwavering support, love, patience, and kindness in the most difficult of times.

Abstract

In this thesis, we explore networks for 5G mobile telecommunication, with a real-time detection of malicious traffic using Deep Learning (DL) and 5G mobile telecommunication testbeds. To investigate the performance of the core network, Software Defined Networking (SDN) and Programming Protocol-independent Packet Processors (P4) were selected due to the potential for programming at the both control and data forwarding layer. SDN and P4 have predominately been researched on an individual basis with limited research combining the two to evaluate improvements to the performance of SDN. We have conducted experiments to explore the hypothesis that combining programmability at both the control plane and data plane provides a platform with better performance in comparison to that achieved with SDN+OvS multi-path, grid and transit-stub network models.

A real-time 5G mobile telecommunication testbed has been constructed combining both software and hardware components. A P4 switch was integrated into the 5G testbed motivated by the performance gains observed in our initial experiments with P4 and OvS switch. Service providers use Multi-access Edge Computing (MEC) technology to provide services on-the-go with low latency, high availability, and high bandwidth, however, MEC nodes are subject to low processing power, which leaves them susceptible to adversaries that may target the platform for malevolent purposes. As a result, we built a 5G testbed that included an MEC node to generate datasets representing both malicious and non-malicious traffic for use in evaluating algorithms intended to detect malicious network traffic.

A new Intrusion Detection System (IDS) has been developed using a 3-layer Convolutional Neural Network (CNN), capable of identifying malicious network traffic. The IDS employs a new injective algorithm capable of encoding network traffic without loss of information as improved RGB images. A separate algorithm capable of decoding RGB images back to network traffic was also developed. The IDS was evaluated in terms of its computational complexity in for example: time, memory and CPU utilisation for the encoding and decoding algorithms, and its accuracy and loss during training and detection. We also applied a Convolutional Neural Network to the dataset created on our testbed and for comparative purposes, to the publicly available datasets UNSW NB-15 and InSDN. The

5G-MEC datasets and detection rate suggest that the employment of current public datasets for research into 5G-MEC security are now inappropriate.

Lastly, we proposed, developed, deployed and evaluated a Real-Time Deep Learning Network Intrusion Detection System (RTDL-NIDS) in an MEC node located in the newly developed 5G-MEC mobile telecommunication testbed in real-time. The deployed Network Intrusion Detection System, conducts a soft real-time detection. The time spent on each detection cycle can be defined as a parameter in the RTDL-NIDS. Hence, this system can be categorised as a soft real-time system. The RTDL-NIDS conducts an initial detection based on known signatures, followed by the encoding of network traffic to images, detection of malicious traffic using our CNN algorithm, and finally decoding of the images to identify the sources of malicious users. We implemented the RTDL-NIDS to function in real-time to collect conclusive results over the application of DL to the intrusion detection problem in 5G-MEC.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms

$2D$	2-Dimensional
$3GPP$	3^{rd} Generation Partnership Project
$4G-LTE$	4^{th} Generation Long Term Evolution
$5G-MEC$	5G-Multi-Access Edge Computing
$5G-SiD$	5G-MEC Signature Detection
API	Application Programming Interface
ARP	Address Resolution Protocol
$bmv2$	Behavioural Model v2
CDN	Content Delivery Network
CN	Core Networks
$CNN3L$	Convolutional Neural Network 3-Layers
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma Separated Values
$DDoS$	Distributed Denial of Service
DL	Deep Learning
DNN	Deep Neural Network

DNS	Domain Name Server
DoS	Denial of Service
DT	Decision Trees
E2E	End-to-End
eMBB	enhanced Mobile Broadband
eNB	Evolved Node B
GoogleColab	Google Colaboratory
GRU	Gated Recurrent Unit
GTP	GPRS Tunnelling Protocol
GUI	Graphical User Interface
HSS	Home Subscriber Server
I2NeT	Image to Network Traffic
ICMP	Internet Control Messaging Protocol
IDS	Intrusion Detection System
IoT	Internet of Things
JSON	JavaScript Open Notation
KPIs	Key Performance Indicators
$KVM-UVT$	Kernal Virtual Machine Ubuntu Virtualised Tool
LLDP	Link Layer Discovery Protocol
LSTM	Long Short-Term Memory
MEC	Multi-Access Edge Computing
ML	Machine Learning
MME	Mobility Management Entity
mMTC	massive Machine Type Communication

NB	Naive Bayes
NeT2I	Network Traffic to Images
NIC	Network Interface Card
NIDS	Network Intrusion Detection System
*NS*3	Network Simulator 3
NSP	Network Service Providers
OAI	OpenAirInterface
OMEC	Open Mobile Evolved Core
ONOS	Open Networking Operating System
OvS	Open vSwitch
*P*4	Programming Protocol Independent Packet Processing
PNG	Portable Network Graphics
QoS	Quality of Service
QUIC	Quick UDP Internet Connections
RAM	Random Access Memory
RAN	radio access networks
RF	Random Forrest
RGB	Red Green Blue
RMSProp	Root Mean Square Propagation
RNN	Recurrent Neural Network
*S*1*AP*	S1 Application Protocol
SCTP	Stream Control Transmission Protocol
SDN	Software Defined Networking
SDR	Software Defined Radio

Sigmoid	"S"-Shaped curve
SIM	Subscriber Identity Modules
$SPGW-C/UandPGW-C$	Control and User plane separation serving gateway
$SSDP$	Simple Service Discovery Protocol
SVM	Support Vector Machines
SYN	Synchronize Packets
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UE	User Equipment
$URLLC$	Ultra Reliable Low Latency Communication
$USRP$	Universal Software Radio Peripheral
$V2V$	Vehicle-to-Vehicle
$VoIP$	Voice over IP

Chapter 1

Introduction

This chapter provides an introduction to my research. The chapter includes research fields, research motivations, research questions, challenges, and contributions to knowledge, that I engaged in during the course of this thesis.

1.1 Research Context

With 4th Generation Long Term Evolution (4G-LTE) completing, and the 5th Generation (5G) mobile telecommunication rollout, more and more service providers are using a Multi-Access Edge Computing (MEC) Infrastructure [2], to provide a faster access time to the content/services by intelligently allocating short-lived, low computational servers, as close as possible to the end user.

The resource allocation, geographical location and computation of MEC have been extensively researched due to the popularity and applicability of MEC. However, despite the advancements in research in the aforementioned fields, the low computational power and on-the-fly realization of MEC create vulnerabilities for Denial of Service (DoS), Distributed Denial of Service (DDoS), Spoof Attacks, and Port Scan attacks. The potential impact of such attacks could create not just service disruptions for multi-media applications under enhanced Mobile Broadband (eMBB), but also disruptions for safety-critical applications that operate under Ultra Reliable Low Latency Communication (URLLC) and massive Machine Type Communication (mMTC).

Malicious intent targeted towards MEC nodes pose a significant threat to MECs, inherently, as they possess low computational resources. MECs can, therefore, potentially be easily exploited by adversaries [3]. Since multiple vendors or organizations use MEC nodes at the edge of the network to offer services or content, this vulnerability may be a threat for multiple organizations. Protecting the MEC launched at the edge of the network, using a built-in Intrusion Detection System (IDS) will result in a higher requirement for computation at the initiation of an MEC edge node. This not only creates a drawback when providing services but also incurs an additional cost to organisations. The complexity of the attacks, and the lack of knowledge required to conduct such an attack, resulted in a higher false negative rate when traditional IDS's were employed, as opposed to a higher true positive rate for IDS backed by an intelligent agent, in the 5G mobile telecommunication architecture [4]. Having a higher chance of detecting spoofing attacks, DDoS, DoS attacks, and Port Scan attacks amidst 5G user traffic which consists of high dimensions and granularity, the application of Deep Learning (DL) towards the detection of malicious traffic has outperformed traditional signature detection methods or the application of Machine Learning (ML) to further the effectiveness of traditional IDS' [5].

1.2 Research Motivations

There are four motivations that have driven this research.

1. Reducing the delay, jitter, loss and congestion in the core network. These will lead towards faster convergence for network packets which in turn will create a direct impact for faster detection of malicious traffic and potentially a higher Quality of Service (QoS) on applications. With the application of programmability at both the control plane and the data plane, the network will be able to handle the heterogeneity of applications, complexity and dimensions of network traffic, without being restricted by a fixed parser, that requires an extension for its specification, each time a new protocol type is introduced. In order to accommodate a protocol that arises with new applications that will occupy the network, the current control plane specification has to be extended manually, whereas, with data plane programmability, this limitation will no longer exist, with the introduction of a flexible parser. The current literature suggests that the application of programmability is only available for one of the above planes (control plane or data plane) but not towards the combination of both. With this combination of technologies (control plane and the data plane programmability), an evaluation of the combination across multiple case studies and technologies has not been undertaken. An evaluation would provide conclusive results over the performance gain by the application of combining programmability in both the control plane and the data plane, to provide a higher QoS and quick detection of intruders.

2. Current literature in the field of 5G and beyond has been developed using simulators or using a desk approach where the research employed a publicly available dataset to

test the effectiveness of an algorithm for security or resource utilisation and allocation. However, the employed datasets were collected on LAN or campus networks, making the application of a tested algorithm, to a real 5G scenario, questionable with a high false positive rate [6]. The aforementioned motivates this PhD research to design and build a 5G testbed and generate datasets from 5G network traffic for the research of 5G and beyond. This is particularly important, due to unique traffic patterns found in 5G telecommunication.

3. The third motivation involves the application of a DL algorithm, Convolutional Neural Network (CNN), in which we develop efficient, faster algorithms to secure the 5G-MEC infrastructure based on the collected dataset from our 5G mobile telecommunication testbed. The current literature has utilised the application of a dataset and with it the data encoding for the CNN algorithm. As CNN accepts images as data for training and testing, data encoding plays a crucial part towards the success of its application. Available literature has utilised 2D-tiled images in Red Green Blue (RGB) or in grayscale. Available data encoding also included mechanisms to cypher network information such as IP addresses and MAC addresses. This information carries important statistics that can be used to identify spoofing attacks. Therefore, the prime focus of our research here revolves around the development of new, improved algorithms for encoding and decoding, and for CNN, and evaluating the computational complexity of these algorithms, which are often overlooked in the literature. For evaluation, the computational complexity of these algorithms was considered. As these algorithms will be executed in the 5G-MEC architecture, which inherently possesses low computation, the resource utilisation of our algorithms are, we believe, crucial.

4. The fourth and final motivation is to apply the newly developed algorithms in real-time to the 5G mobile telecommunication testbed and to evaluate the performance of the algorithms. Therefore, the new algorithms for encoding were implemented in the 5G mobile telecommunication testbed. With this, a challenge was derived. That was, how to collect network traffic and conduct quick signature-based detection, given that the resulting suspicious network traffic can be used to create RGB images. The images are to be sent to the CNN algorithm in real-time, for a true two-staged detection [7] using a CNN algorithm, where the algorithms have been implemented in a 5G mobile telecommunication testbed. The aforementioned indicates that by employing a multi-staged detection process, the application of CNN can be implemented in real-time to a 5G mobile telecommunication testbed.

1.3 Research Aim and Objective

This study seeks to safeguard the 5G-MEC infrastructure on the periphery of a 5G-MEC mobile telecommunications testbed with data plane programmability, by implementing a two-stage intrusion detection system augmented with deep learning. The low computational requirement of the newly developed algorithms translates to securing the MEC edge nodes without utilising additional resources.

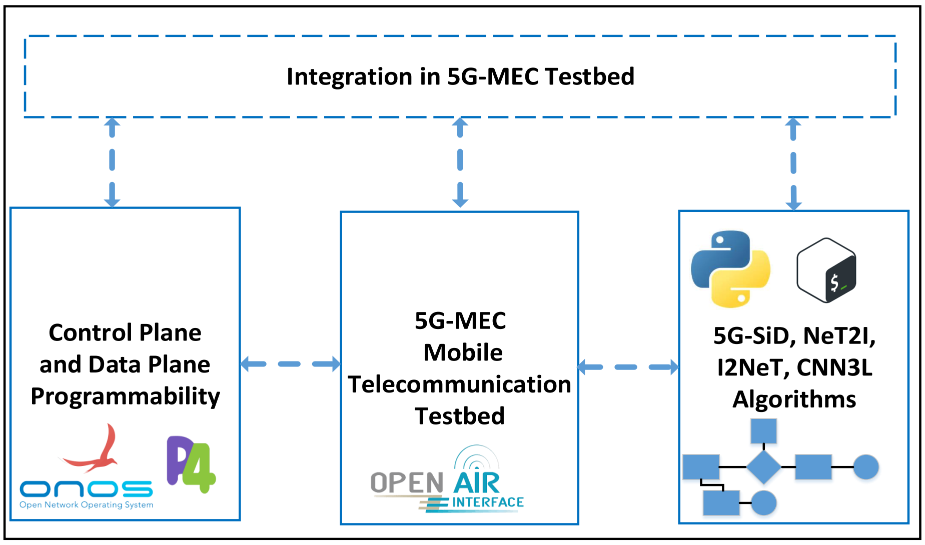

Integration in 5G-MEC Testbed
Control Plane and Data Plane Programmability
ONOS
Open Network Operating System
P4
5G-MEC Mobile Telecommunication Testbed
OPEN AIR INTERFACE
5G-SiD, NeT2I, I2NeT, CNN3L Algorithms

Fig. 1.1 Overview of the thesis

In order to achieve this, the research was organised as depicted in Fig.1.1. They can be described as follows:

1. To evaluate the underlying switching mechanisms which involve, programmability in the control plane against programmability in both the control and data plane. Multiple case studies and topologies were considered.

2. To design and build a 5G mobile telecommunication testbed with a data plane programmed switch (Objective 1) to further the research of 5G and to aid in the creation of a dataset suited for 5G telecommunication networks.

3. To Propose a new encoding mechanism (Network Traffic to Images (NeT2I)), as opposed to greyscale or 2D x and y dimensional images, which translates network information into RGB images with lower computational complexity. To Propose a new detection algorithm to test the nature of the traffic (malicious or non_malicious) by employing a CNN3L algorithm. To evaluate the algorithms developed on publicly

available datasets. To propose a new decoding mechanism (Images to Network Traffic (I2NeT)), which translates RGB images to network traffic with lower computational complexity.

4. The development of a Real-Time Deep Learning Network Intrusion Detection System (RTDL NIDS) by streamlining the algorithms (Objective 3) in real-time on the 5G mobile telecommunication testbed (Objective 2). Proposal of a new early signature detection algorithm (5G-SiD) with a lower computational complexity in comparison to two other popular NIDS, to generate a dataset for the NeT2I (Objective 2), and upload them to the CNN3L detection algorithm (Objective 3), to evaluate the nature of the traffic, and finally an affirmation of the nature (malicious or non_malicious) by using I2NeT.

1.4 Research Methods

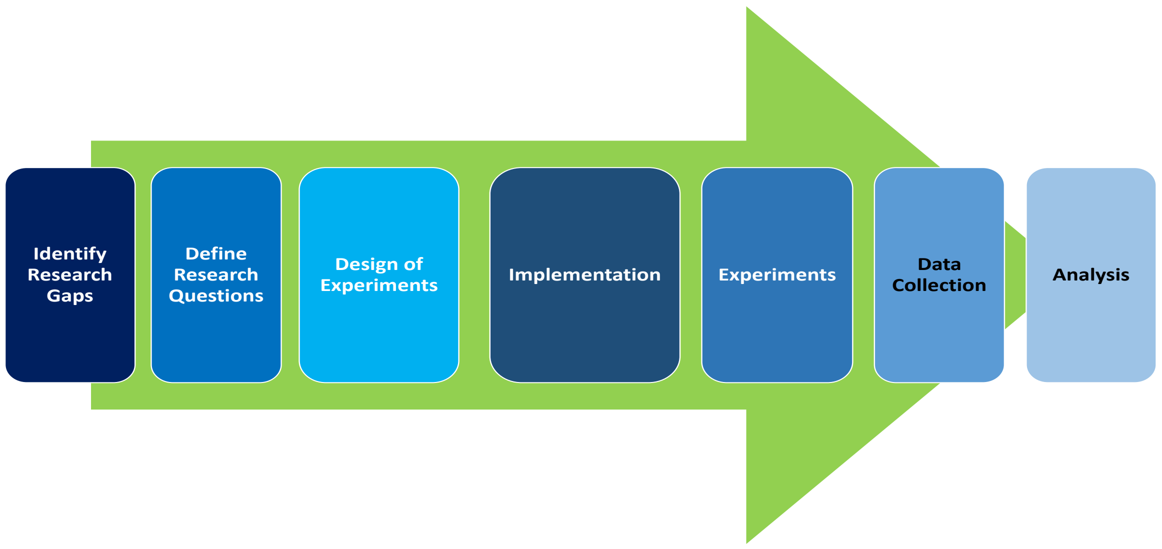


Fig. 1.2 Depiction of the research method employed for this thesis

This research has been organised in the following manner as depicted by Fig.1.2. Initially, the gaps in the literature will be presented, followed by the design of experiments based on research questions (section 1.6), the implementation of experiments, and the collection and analysis of results from our experiments.

1.5 Research Gaps

1.5.1 Research Gaps Identified from the Literature Search

The literature search was carried out in the following order. Improvements to the underlying core network to reduce delay, jitter, congestion and loss and thereby improve the performance of the core network to provide a higher QoS and faster convergence of network packets for faster delivery. This also aided in the quick detection of malicious network traffic. Next, the literature focused on 5G security, particularly with respect to the desk application of an algorithm for a dataset. This helped to establish the importance of employing a testbed to conduct research in 5G and beyond. Next, the literature based on the encoding/decoding of data towards a DL algorithm, CNN will be presented. With it, the literature based on the DL algorithms and the accumulated advantages for the 5G IDS problems will also be presented. Finally, the literature for a two-staged detection of malicious intent will be presented, finalising the literature review, which motivated this study. The order in which the literature search was carried out is depicted in Fig. 1.3.

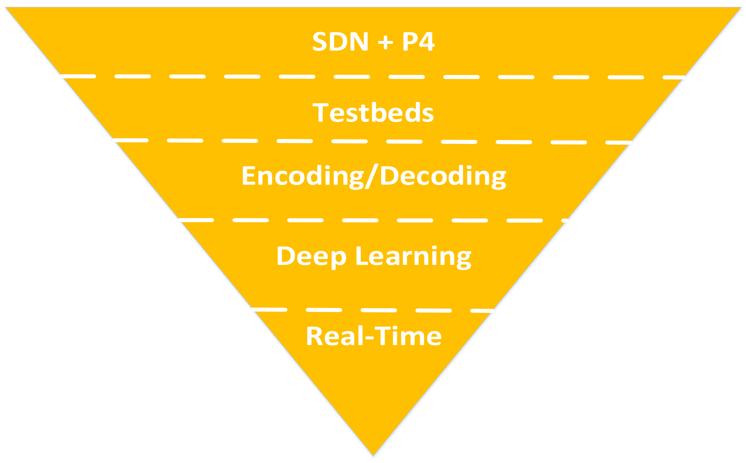


Fig. 1.3 Literature review of this study and the organisation of material.

The gaps that can be drawn from the literature search can be listed as follows.

1. The need for reducing the delay in the core network in order to improve the performance of the applications, that occupy the network. Improve the performance of the network for faster packet convergence and thereby detect malicious traffic quickly.

   Research Challenges

   (a) Available emulation tools that can facilitate and emulate programmability in both the control plane and data plane.

   (b) Compatibility of tools and software with the emulation tool that can facilitate programmability in the control plane and data plane.
   (c) Researching network topologies in the existing literature that can be replicated.

   Contributing to the aforementioned research gap, the following solutions were achieved upon researching the challenges. Employment of Mininet as the emulation tool, ONOS as the network controller, OvS and BMv2 for achieving programmability at the control plane and data forwarding plane, employment of the most popular traffic types to emulate a network to achieve realism, and emulating three network topologies that are found in the literature.

2. The need to employ a testbed to further the study and the evaluation of 5G research.

   Research Challenges

   (a) Available tools, equipment, and software to launch a testbed.
   (b) Installation, testing, and launching the testbed.
   (c) Study of the existing datasets used for intrusion detection research and the creation of a dataset based on network traffic from the testbed.

   The following solutions were accomplished in achieving the research gap amidst the aforementioned challenges. OpenAirInterface was chosen as the testbed to launch a 5G mobile telecommunication testbed. A dataset was created that possesses traffic patterns unique to a 5G mobile telecommunication network after studying popular datasets, UNSW NB-15 and InSDN.

3. The need for the employment of DL as opposed to ML, for the intrusion detection problems of 5G and beyond.

   Research Challenges

   (a) Reviewing existing research on Intrusion Detection based on Machine Learning and Deep Learning.
   (b) Study of performance in terms of the detection rate for Machine Learning and Deep Learning algorithms.
   (c) Study of Convolutional Neural Network and the implementation of an algorithm.
   (d) Study of the existing encoding and decoding algorithms and the associated computational complexity.

Upon studying the existing literature and research on ML and DL, DL was chosen as opposed to ML. From DL algorithms, CNN was chosen as the detection algorithm with 3 layers of convolutions due to the higher accuracy achieved by them as opposed to other DL algorithms. Also, the RGB encoding mechanism was considered as opposed to grayscale images, given the higher accuracy achieved by them.

4. The use of algorithms in the 5G-MEC and their computational complexity.

   Research Challenges

   (a) Study of the computational capabilities and limitations in MEC nodes.
   (b) Collection of data pertaining to computational complexity from the execution of an algorithm.

   Upon studying the available functions and libraries to collect computational complexity, the following libraries were chosen. Time, memory_profiler, and psutil were chosen to collect computational complexity. The collated statistics were collected on an MEC node launched using UVT-Cloud environments to ensure that the algorithms can be executed in MEC nodes.

5. Two staged detection of malicious traffic for 5G and beyond using an intelligent agent.

   Research Challenges

   (a) Study of the existing hypothetical two-staged Intrusion Detection Systems.
   (b) Researching tools, software, and libraries required to implement algorithms capable of conducting a two-staged detection.
   (c) Programming a continuous execution of an algorithm in a computationally viable manner.
   (d) Connecting an intelligent agent launched in the Google Cloud to the local MEC.

   The following solutions were created to address the challenges. Studying the existing real-time hypothetical NIDSs' and implementing a workflow for implementation. Employing tools and technologies such as Pyshark, shell scripting, and various Python libraries for conducting a two-stage detection.

6. The need for the real-time application of DL-based IDS system.

   Research Challenges

(a) Non-existence of soft real-time detection mechanisms that utilise an intelligent agent for intrusion detection.

(b) Study of tools, technologies and software that can be utilised towards the creation of a soft real-time detection mechanism.

Implementation of a soft real-time NIDS that can conduct intrusion detection using signatures and deep learning. Use of tools such as Google Colab and TensorFlow were used extending the two staged detection to be a soft real-time NIDS.

1.6 Research Questions

Computer scientists are inherently expected to conduct more and more research and experiments [8, 9], primarily to obtain a consistent result or a collection of results that can only be obtained with a repeated process involving a collection of feasible experiments [10]. The primary method of research for this thesis was conducted employing a series of quantitative methods, derived from data collected from the designed experiments. The experiments were designed with the research questions in mind. From a user's perspective, receiving a service with less congestion, loss and delay will be a binary finding, as the service has minimal to no downtime or the service suffers unavailability. From a service provider's perspective, the service has to be provided during normal conditions and during an attack. The service provider will also be able to detect malicious traffic at a faster speed since this research is aimed at providing the faster delivery of network packets and the detection of maliciousness using a two-staged detection mechanism involving a DL component.

RQ1: How does programmability in both the control plane and data plane affect network performance?

I researched the existing literature with the intention of improving the performance of the underlying network. Improvements to the underlying network were either focused on a particular case study or an application. Given the heterogeneity of devices, the complexity that arises from the use of various applications and the congestion that arises with the exponential growth of connected users and devices, it was preferable to evaluate the network as a whole, in order to identify the best packet forwarding or processing mechanism. This is important for the following three reasons: 1: in order for the service providers to provide the best QoS to end users, 2: a network with minimal delay and latency with a higher throughput is the most desirable avenue to consider, and finally, 3: when there's malicious intent occurring in the network albeit, from an external or an internal source, a quick detection will alleviate performance degradation.

In order for quick detection to occur, faster transmission of network packets is a crucial factor to consider. As the Internet grows towards 5G and beyond, many applications and vendors will occupy the network space and with it, there is a need for extending the specification of the control plane, for the new network traffic to pass through or to be processed. In order to diminish the constriction of extending the specification for packet processing, a parser independent from the target switch and protocol with capabilities to extend and reconfigure in future, should the need arises, is the most ideal solution to future networks that has the capability to implement data plane programmability and to process network packets in parallel as opposed to sequential processing. Given the faster processing capabilities of packets, not only this solution will aid the networks in providing higher performance, but will also aid in the quick detection of malicious traffic.

RQ2: How do we design and build a testbed for 5G mobile telecommunications?

Current literature employs datasets to demonstrate the accuracy of the algorithm in research for 5G security. There is a question regarding the validity of the datasets and the traffic patterns found in them.

From a security perspective frameworks/applications for 5G have been presented in the literature, however, these studies do not include testbeds for their respective work but rather present frameworks based on literature for specific use cases (eMBB, URLLC, and mMTC). The inclusion of testbeds will, I believe provide a deeper understanding of the feasibility of the frameworks/applications. To accomplish the realisation of a testbed to further the conclusiveness of results, I researched the available technological advancement in 5G to launch a testbed capable of demonstrating a 5G mobile telecommunication, maintaining compliance with the 3GPP standards for both core networks (CN) and radio access networks (RAN).

RQ3: How can we design new algorithms for intrusion detection using a DL agent in 5G-MEC?

For securing 5G networks and Multi-access Edge Computing (MEC) infrastructure, Deep Learning (DL) is a viable candidate for the problem of detecting malicious traffic. The ease of training and generalisation, in comparison to other fully connected networks together with the high accuracy rate achieved with Convolutional Neural Networks (CNN), has prompted us to explore CNN as a viable option for the detection of malicious traffic in the 5G network infrastructure.

Despite the high accuracy produced by CNN as opposed to other DL and ML algorithms, a challenge exists when converting network traffic (integer and string type data) to a form (images) recognisable by a CNN algorithm. CNN accepts images in both grayscale and RGB forms. Since grayscale has limited colour space to represent information, RGB colour space is the prominent option to consider.

Since the MEC infrastructure will possess low processing power, a CNN-based IDS must produce images for the detection algorithm requiring less computation. Research-based on encoding network traffic into images has not discussed computational complexity in the application of their respective algorithms. Similarly, the computational complexity of decoding, encoded images back to network traffic should also produce less computation, due to the low processing power of MEC nodes. Current research has not produced decoding algorithms for their respective encoding algorithms, nor evaluated their computational complexity.

RQ4: How do we apply DL in real-time to intrusion detection in 5G-MEC mobile telecommunication testbed?

Previous studies on the application of CNN towards NIDS have been focused on testing the effectiveness of a desired feature selection or DL algorithm by employing a dataset/s. However, in order to test the hypothesis that the application of DL will be beneficial for NIDS on 5G networks, has to be tested by employing 5G datasets collected on a 5G mobile telecommunication testbed. Further to the employment of 5G data, the application of employing an intelligent agent such as a DL should be advanced from the traditional desk approach. This is because, an algorithm trained and tested using a desk approach (off-line mode), may not function correctly in real-time. By applying a DL algorithm in a real-time 5G mobile telecommunication testbed, we can attest to the generated accuracy as opposed to the accuracy presented by a desk approach research.

To reduce the computation by the detection algorithm (CNN), a quick detection based on signatures, with less computational power is beneficial. This method alleviates the pressure on the CNN-based detection algorithm, as the signature-based detection algorithm will be able to filter and remove non_malicious traffic from progressing to the CNN-based detection. A multi-staged detection mechanism has been theoretically presented in the literature, however, they have not been implemented. The implementation of a multi-staged NIDS with a DL element, on a 5G-MEC mobile telecommunication testbed, will we believe provide conclusive results to the applicability of DL towards securing a real 5G-MEC network.

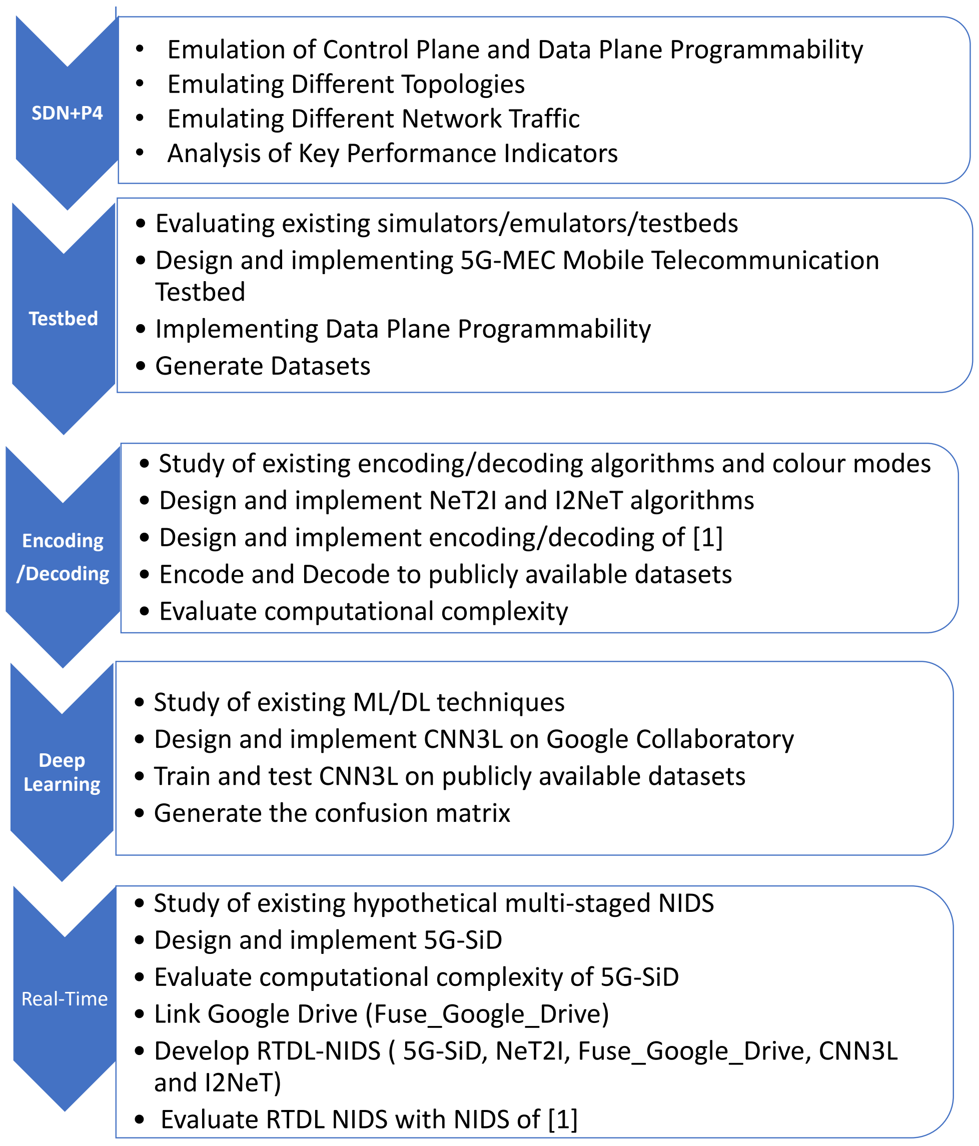

SDN+P4
• Emulation of Control Plane and Data Plane Programmability
• Emulating Different Topologies
• Emulating Different Network Traffic
• Analysis of Key Performance Indicators
Testbed
• Evaluating existing simulators/emulators/testbeds
• Design and implementing 5G-MEC Mobile Telecommunication Testbed
• Implementing Data Plane Programmability
• Generate Datasets
Encoding /Decoding
• Study of existing encoding/decoding algorithms and colour modes
• Design and implement NeT2I and I2NeT algorithms
• Design and implement encoding/decoding of [1]
• Encode and Decode to publicly available datasets
• Evaluate computational complexity
Deep Learning
• Study of existing ML/DL techniques
• Design and implement CNN3L on Google Collaboratory
• Train and test CNN3L on publicly available datasets
• Generate the confusion matrix
Real-Time
• Study of existing hypothetical multi-staged NIDS
• Design and implement 5G-SiD
• Evaluate computational complexity of 5G-SiD
• Link Google Drive (Fuse_Google_Drive)
• Develop RTDL-NIDS (5G-SiD, NeT2I, Fuse_Google_Drive, CNN3L and I2NeT)
• Evaluate RTDL NIDS with NIDS of [1]

Fig. 1.4 Milestones of this PhD

1.7 Research Milestones

Fig.1.4 depicts the milestones of this research. We initiate our research on an emulated environment where we evaluated the applicability of achieving programmability in both the control plane and data plane. Having achieved the above, we moved on to the design and the implementation of a 5G-MEC testbed for the creation of datasets to advance security-based research on 5G and beyond. Having created datasets which resembled the unique traffic patterns of 5G mobile telecommunication, we designed and developed new algorithms for the detection of malicious network traffic in 5G-MEC. This included encoding/decoding algorithms and a CNN-based detection algorithm. Our final research milestone is the application of the previously mentioned algorithms to the 5G-MEC mobile telecommunication testbed, as a form of a multi-staged NIDS. As depicted in Fig.1.4, each of the aforementioned milestones will be discussed in-depth in a dedicated chapter in the following manner.

- Chapter 3: A Performance Evaluation for Software Defined Networks with P4
- Chapter 4: A Real-Time 5G Mobile Telecommunication Testbed
- Chapter 5: New Algorithms for the Detection of Malicious Traffic in 5G-MEC
- Chapter 6: Real-Time Application of Deep Learning Intrusion Detection in 5G-MEC

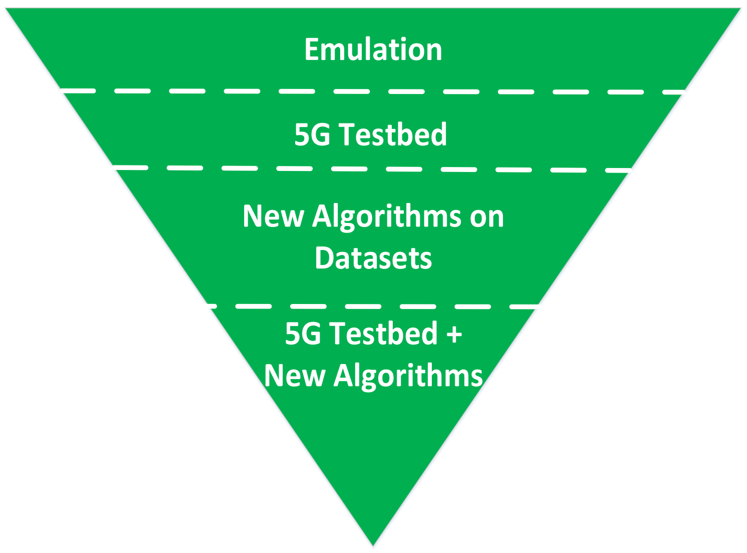


Fig. 1.5 Implementation of Experiments.

1.8 Implementation

This section refers to the implementation of the experiments outlined in the previous section. Fig.1.5 depicts the manner in which implementations of the experiments were conducted. Our initial research was carried out using an emulated environment to evaluate the benefits of combining the control plane and data plane programmability. Mininet utilised a Python API to specifically emulate networks, devices, and links. Having completed emulations, we designed and developed a 5G testbed composed of User Equipment (UE), a RAN and a CN, which is composed of entities that one would find in a mobile telecommunication network. To incorporate MEC into the 5G mobile telecommunication testbed, Ubuntu UVT Cloud environments were employed. Next, as per Fig.1.5, we designed and developed new algorithms for the detection of malicious traffic in 5G-MEC. The developed algorithms were tested and evaluated on publicly available datasets. Finally, we implemented the newly designed and developed algorithms on the developed 5G testbed, to function in real-time to detect malicious network traffic. Further details of the implementation of experiments can be found under each respective chapter.

1.9 Contributions

My research has been conducted in a manner where the main contributions can be summarised into two main sections. In the former, I focused on mitigating the delay in the core of the network that may help in achieving the Key Performance Indicators (KPIs) for 5G. Upon realisation of the application of *SDN+P4* the most promising solution to overcome the delay in the core of the network, I implemented a mobile telecommunication testbed that can be used to implement an MEC distribution using UVT-Cloud environments, create datasets, extract features in real-time, encode network traffic exceeding 24-bits into RGB space. The following presents a detailed description:

1. Evaluated the performance of networks when *SDN+P4* is employed rather than *SDN+OvS*.

    - The research found that initialising *SDN+P4* with parallel processing of packets, improves performance on applications in comparison to *SDN+OvS*, which uses sequential processing.
    - Evaluated the quality of applications due to faster processing achieved with *SDN+P4* in comparison to *SDN+OvS*. The statistics such as increased bps and

throughput, reduced delay jitter, packet loss, delay and buffering time have led to a higher quality of application at the receiver's end.

- Investigated the overhead created due to the slow path utilisation of OvS and the performance variation in comparison to a P4 target switch. With the evolution of the internet and the increased number of connected devices, networks will face congestion. With more and more packets requiring processing using a controller, a network model that utilises a slow path approach such as OvS will potentially lead to an exponential growth in traffic congestion.

2. A Testbed was developed employing functions of a 5G-MEC mobile telecommunication network with fully functioning P4-BMv2 switches in the Radio Access Network and Core Network.

   - A dataset was generated from the testbed containing both legitimate and malicious types of traffic found in 5G networks. This dataset can help in mitigating the False Positive rate for an Intrusion Detection System, applied to a 5G mobile telecommunication network.
   - The dataset was tested and compared to existing datasets not generated from a 5G testbed.

3. A new algorithm to encode network traffic, for example, IP addresses, and MAC addresses to RGB Images and a new algorithm to decode, encoded RGB images into network traffic.

   - A new IDS using CNN and the proposed encoding and decoding algorithms for the detection of malicious network traffic.
   - Evaluation of the proposed IDS in terms of computational complexity in, for example, time, memory and CPU utilisation, together with accuracy and loss in training, validation and detection.
   - Comparison of the proposed IDS against a significant IDS that uses a different approach for encoding and CNN detection.

4. A new Real-Time Deep Learning based Network Intrusion Detection System (RTDL-NIDS)

   - A novel signature detection algorithm to conduct a quick signature-based detection (5G-SiD) in the 5G-MEC mobile telecommunication testbed.

- A new RTDL-NIDS launched in the 5G-MEC mobile telecommunication testbed for real-time detection of malicious network traffic using CNN
- Evaluation of the proposed RTDL-NIDS in terms of computational complexity in, for example, time, memory and CPU utilisation, together with accuracy and loss in training, validation and detection.
- Comparison of the proposed RTDL-NIDS against a significant NIDS that was implemented in the testbed, that uses a different approach for signature-based detection and encoding.

1.10 Publications

- Fernando, Omesh A., Hannan Xiao, and Xianhui Che. *"Evaluation of Underlying Switching Mechanism for Future Networks with P4 and SDN (workshop paper)."* International Conference on Collaborative Computing: Networking, Applications and Worksharing. Springer, Cham, 2019.
- Fernando, Omesh A., Hannan Xiao, and Joseph Spring. *"Developing a Testbed with P4 to Generate Datasets for the Analysis of 5G-MEC Security."* 2022 IEEE Wireless Communications and Networking Conference (WCNC). IEEE, 2022.
- Fernando, Omesh A., Hannan Xiao, and Joseph Spring. *"New Algorithms for the Detection of Malicious Traffic in 5G-MEC."* 2023 IEEE Wireless Communications and Networking Conference (WCNC). IEEE, 2023.
- Fernando, Omesh A., Hannan Xiao, Joseph Spring and Xianhui Che. *"An Evaluation of Software Defined Networking"* 2023 IEEE Transactions on Networking, 2023. (Submitted)
- Fernando, Omesh A., Hannan Xiao, and Joseph Spring. *"Real-Time Application of Deep Learning to Intrusion Detection in 5G-Multi-Access Edge Computing"* 2023 IEEE Transactions on Machine Learning in Communications, 2023. (Submitted)

1.11 Structure of my Thesis

My dissertation is organised as follows.

Title	Type	Cited by	Year
Real-Time application of Deep Learning to Intrusion Detection in 5G-Multi-Access Edge Computing (Submitted)	Journal Article		2023
A Performance Evaluation for Software Defined Networks with P4 (Submitted)	Journal Article		2023
New Algorithms for the Detection of Malicious Traffic in 5G-MEC	Conference Paper		2023
Developing a Testbed with P4 to Generate Datasets for the Analysis of 5G-MEC Security.	Conference Paper	4	2022
Evaluation of underlying switching mechanism for future networks with P4 and SDN	Conference Paper	4	2019

Table 1.1 Publications and the citation at the time of submitting the PhD Thesis

- **Chapter 2** provides a detailed collection of background information on topics such as Software Defined Networking, Programming Protocol independent Packet Processing, Implementation of the testbed and other simulators/emulators employed in literature, Deep Learning and its advantages, Multi-Access Edge Computing and lastly, the application of Deep Learning towards the intrusion detection problem.

- **Chapter 3** evaluate the SDN with the application of P4. This chapter, explains the need for control plane programmability and the importance of combining the paradigm of programmability with the data plane. This chapter was aimed at evaluating the performance of *SDN+P4* which combined the programmability at both the control plane and data plane as a means of achieving better performance by reducing delay, jitter and loss in the core network. In this chapter, we also evaluated if particular traffic outperforms other protocols. This was done primarily because the future application will require faster convergence of traffic due to stringent requirements.

- **Chapter 4** introduces the real-time 5G mobile telecommunication testbed that encompasses an MEC deployment. It was developed in order to collect a dataset containing 5G mobile telecommunication traffic and evaluate the dataset against well-known publicly available datasets. The chapter also presents the training and validation accuracy along with the loss associated with it for both the collected dataset and a publicly available dataset.

- **Chapter 5** presents the new algorithms that were developed towards encoding collected network traffic data into RGB images. The algorithm, NeT2I encodes network traffic from a CSV file to a collection of PNG images whilst the I2NeT algorithm decodes the PNG images into creating a CSV file. Furthermore, this chapter also presents the CNN3L algorithm which is capable of binary classification between malicious to non-malicious network traffic. In this chapter, the newly developed IDS was evaluated with a comparative IDS. The comparative IDS was designed to benchmark a popular encoding algorithm to test the effectiveness of the newly developed IDS. As opposed to training and validation accuracy together with the loss, computational complexity was also presented, since the application of the proposed IDS can be deployed in a low processing power MEC node.

- **Chapter 6** introduces the RTDL NIDS. This new NIDS collect network traffic at the edge of the 5G-MEC mobile telecommunication network conducts feature selection, effectuates a signature-based detection (5G-SiD) of known malicious traffic patterns, enables the NeT2I algorithm to encode network traffic to PNG images, and lastly uploads the images to the predict function of the CNN3L detection algorithm, decode the images using I2NeT. RTDL NIDS was designed and implemented to conduct detection in real-time. With the employment of the RTDL NIDS, an MEC node capable of carrying out NIDS based on DL can be implemented at the edge. In this chapter, matrices including time, CPU and RAM were also presented to evaluate the performance of the new NIDS against the NIDS of [1]. The NIDS from [1] was also streamlined as above, to evaluate the performance of RTDL-NIDS. Accuracy and loss were also presented for both NIDSs that were accumulated during training and validation.

- **Chapter 7** concludes this dissertation by discussing the overall achievements and contributions. It also presents the limitations and our future research work.

1.12 Chapter Summary

The above chapter has provided an overview of the dissertation. The initial section provided an introduction to the topics that were studied in this research, which formulated the research questions and resulted in the contributions following the publications. In the next chapter, we present the literature which motivated this thesis.

Chapter 2

Literature Review

2.1 Overview

This section presents the findings from the literature search that was carried out, during the development of this thesis. Each section will present an introduction followed by a related work section. The literature listed below has been referenced at later stages throughout this thesis.

Section 2.2, introduces the literature pertaining to the core of the network. This section is important to reduce the delay, loss and jitter in the core network for faster convergence of network traffic together with faster detection of malicious traffic. Section 2.3, presents the literature that we utilised for the development of a 5G mobile telecommunication testbed. This section also refers to the existing literature that has utilised either a desk approach or a simulator for research in the realm of 5G Security. Finally, Section 2.4, highlights the literature search carried out for the development of the algorithms.

2.2 A Core Network using SDN + P4

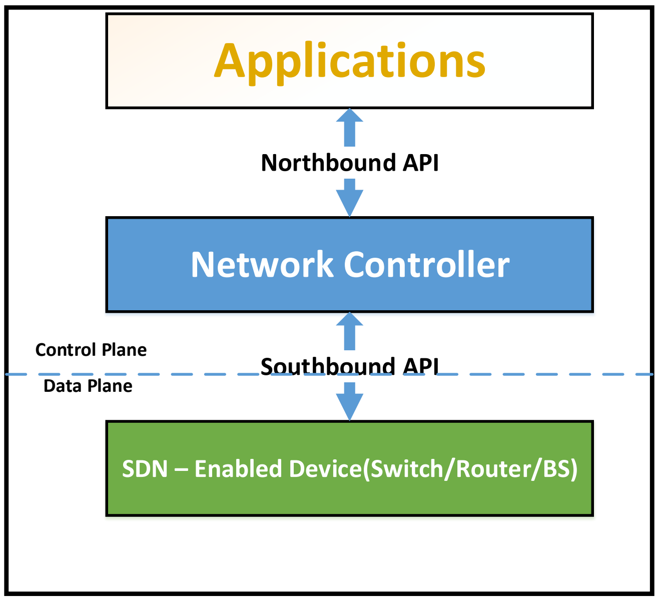
Software Defined Networking (SDN) [11, 12] allows for the decoupling of the control plane from the data forwarding plane facilitating an improved performance via programmable network management. The flexibility obtained enables virtualisation, and centralised control for the network, with routing tables being generated and updated centrally by a network controller. The separation of the control plane and data plane is made possible through a clearly defined interface which connects the switches to the network controller. This interface employs an Application Programming Interface (API), which allows the controller to exert direct control over the data plane elements (networked devices). The protocol OpenFlow

[13, 14],is a notable example of such an API. Fig.2.1a highlights the functionality of the network controller and OpenFlow.

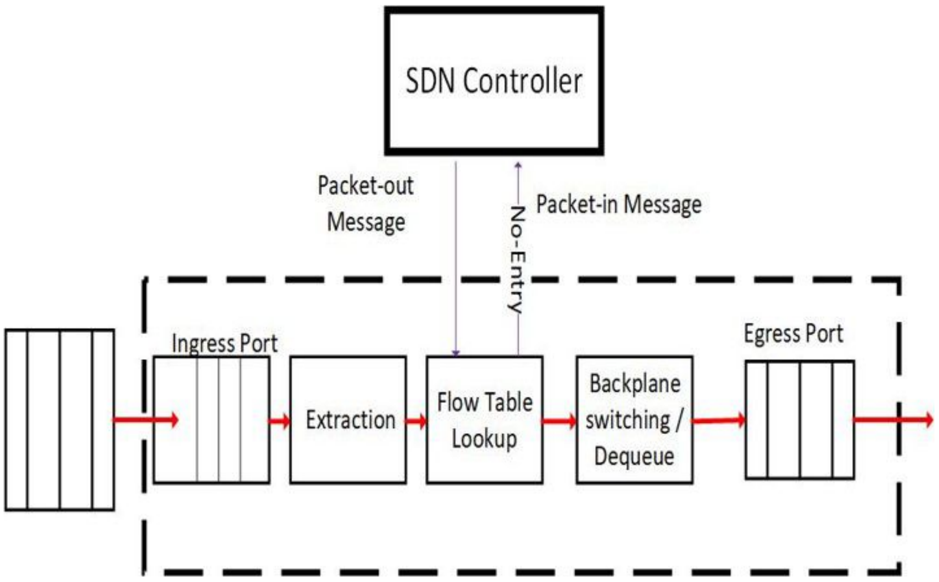
An OpenFlow forwarding rule consists of a ‘match’ and ‘action’. The ‘match’ involves the matching of packet header fields such as source and destination fields and an ‘action’ involves the action to be performed such as forward or drop a packet. This function is referred to as ‘match+action’ [15]. The controller installs these flow entries in the Flowtables of the SDN enabled devices. If a matching flow entry is found against a packet in the data flow, the predefined action (pass or drop) for that entry is performed on the matched packet. If no match is found, the packet is forwarded to the network controller as a Packet-in Message. The controller is responsible for determining how the packet should be handled by returning this specific packet to the switch and stating which port it should be forwarded to (Packet-out message). This has been illustrated in Fig. 2.1b.

OpenFlow was first introduced with 12 ‘match’ header fields [16], and now stands at 44 header fields, as described in the OpenFlow specification 1.5.1 [14]. These are anticipated to grow exponentially in the foreseeable future, in response to the evolution and heterogeneity of the internet. This could potentially be a problem as the application of new header field extensions cannot be achieved when using a stringent-fixed parser at run time, whilst the switches are actually processing packets.

An alternative to extending the OpenFlow specification is to employ a novel parser that can carry out ‘match+action’ in parallel as opposed to sequential allowing for programmability in the data forwarding plane without having to modify the OpenFlow specification. In our abstract model, as depicted in Fig. 2.2a, switches forward packets via a programmable parser followed by multiple stages of match+action, arranged in series, parallel or a combination of both. The forwarding model is controlled by two types of operations: Configure and Populate. The configure operation (which is composed of Parse graph, Control Program, Table configuration, and Action set), programs the parser, arrange the order of match+action, and specify the order of fields processed by each stage. The populate operation, adds, or removes entries to the match+action tables that were specified during configuration. Such a parser can be updated, extended and modified without interrupting packet processing and forwarding and would allow for heterogeneous applications. A protocol that can be employed to carry out ‘match+action’ in parallel is the Programming Protocol Independent Packet Processing (P4) [16]. Fig. 2.2b, shows the relationship between P4 and a switch, telling the switches how packets are to be processed using existing APIs (such as OpenFlow) that are designed to populate the forwarding tables in switches. P4 raises the abstraction for programming the network and can serve as a general interface between the controller and the



(a) Software Defined Networking (SDN).



(b) How SDN is implemented

Fig. 2.1 Functionality of SDN

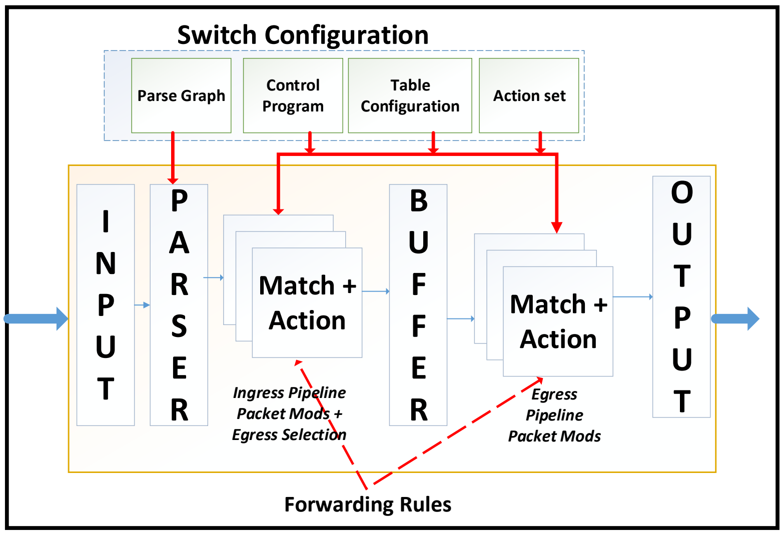
switches. That is, we believe the future of OpenFlow, where the controller tells the switch how to operate, rather than be constrained by a fixed switch design.

P4 has three main properties: re-configurability, protocol independence, and target independence allowing the network administrator to determine the functions and capabilities of a switch rather than adhering to vendor's specification [17]. Utilising a common open interface, the administrator can leverage P4 to program a flexible parser to match new header fields as opposed to working with a fixed parser in OpenFlow which would require a specification update in order to process new header fields. An illustration of P4 functionality and its interaction with a target switch can be found in Fig.2.2.

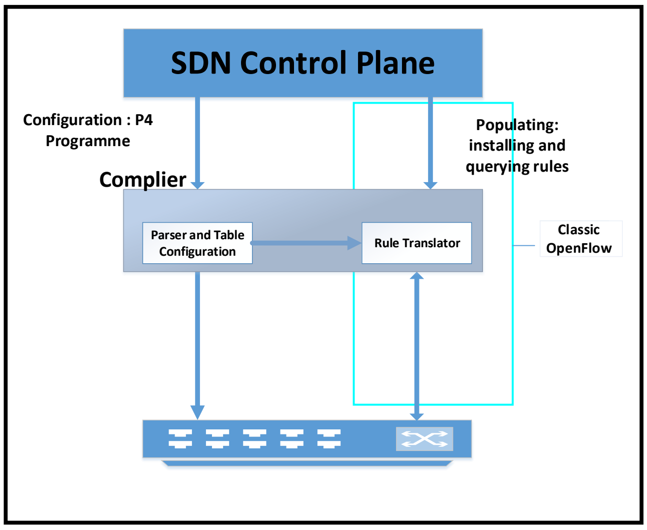
The extensive research carried out in [18, 19] indicate that the majority of future network traffic is expected to be the result of using hand-held smart devices. It is stated that the increased traffic due to hand-held devices has resulted in an approximately 82% [19] of all consumer Internet traffic. The increased load of end-user IP traffic threatens to lead to network congestion resulting in a reduced Quality of Service (QoS) in terms of, for example, jitter [20], packet loss, and throughput [21] with service providers trying to maintain an acceptable level of service. It is interesting to note a challenge presented in [22], which suggests that the majority of the delay in data communication occurs at the core of the network due to an exponential growth in periodic updates [23] used to maintain the network state.

Control and data plane programmable functions motivate us to investigate how this emerging approach can improve the performance of a network by processing packets faster. It is, we feel, crucial to investigate whether and how the extent to which a solution with programmability in both the control plane and forwarding plane will provide an improved core network to accommodate stringent performance and flexibility when network traffic increases continuously.

Research in [24] discussed technological enhancements for networks highlighting technologies such as OpenFlow, P4, the Data Plane Development Kit and Click-based solutions. It was established that the protocol-independent nature of the high-level language P4 [16] together with independence from the underlying hardware and header limitation benefits the network, increasing performance through the faster processing of packets. The survey [24, 25] compliments the research of [16] on P4. The utilisation of the language P4 has been presented for many applications. [17, 26–32]. Research in [17] applied the programmability of P4 to create an application capable of handling data centre traffic. In [26] a congestion control mechanism has been used together with P4 and in [27] mechanisms for packet processing capabilities through P4 have been utilised with a Robust Header Compression (ROHC) scheme for improving performance. Further examples include a tool



(a) Programming Protocol Independent Packet Processing (P4).


SDN Control Plane
Configuration : P4 Programme
Complier
Populating: installing and querying rules
Parser and Table Configuration
Rule Translator
Classic OpenFlow

(b) How P4 and classic OpenFlow is implemented with SDN

Fig. 2.2 Functionality of P4

for developing and evaluating data plane applications [28], Service Function Chaining on P4 devices in [29] and in [30], an extension to the application of Open vSwitch, to act as a hypervisor switch.

A study conducted in [33] employed an identical topology to that initially presented in [34] which highlights the validity of the respective initial work. The study focused on economic factors from a hypothetical migration towards OpenFlow for Network Service Providers (NSP) but did not evaluate network performance. In [35] and [36] a methodology to orchestrate a dynamic end-to-end (E2E) path between transit stub domains was presented, with the idea of SDN successfully managing traffic between NSP. How an SDN controller should be placed between NSPs was discussed in [37] as a means of improving performance. This was achieved through the ‘correct’ and optimal placement of SDN controllers at various topological locations. A similar approach has been employed in [38] where a second controller was placed to mitigate traffic overload in the core network. In [39] it was suggested that in order to achieve better performance, a minimum of 20% of the nodes in the core network should operate as SDN controllers with control plane programmability. In [40] and [41] the authors utilised control plane functions in an SDN controller to improve performance. Similarly, [42] presented that when faced with congestion SDN networks face performance degradation. In addition, in [43] the use of a high-level specification template for management patterns was utilised to improve performance in SDN networks.

Various researchers have utilised the P4 language to improve the performance of a particular case study or an instance. To the best of our knowledge research aimed at improving the performance of a network through the utilisation of programmability at the control plane and data forwarding plane (*SDN+P4*) have been limited to improving performance for specific case studies only.

Therefore, we hypothesise that a combination of data plane and control plane programmability (*SDN+P4*), as illustrated in Fig.2.1b will elevate the performance of the network, to accommodate stringent performance requirements for future applications in comparison to ‘Open vSwitch (OvS) [44] coupled with control plane programmability’ (*SDN+OvS*). The two models can be seen in Fig.2.2b.

2.3 Real-Time 5G mobile telecommunication Testbed

5G mobile networks will become the key enabler and foundation for Information Communication Technology, catering to a diverse set of use cases (enhanced mobile broadband, massive machine type communication and ultra-low reliable low latency communication) with a range of different requirements. Providing support to each of the use cases using a common architecture has led to a significant change in design philosophy for core and radio access networks, and the application of Multi-access Edge Computing (MEC) involves a significant modification at the service based architectural level in order to (potentially) cater for the diverse use cases involved.

Multi-access Edge Computing and methods of deployment have been presented in [2] and as awareness and interest in 5G-MEC has grown within the industry and academia, research based on resource allocation, energy awareness and network slicing has also grown. Flexibility, scalability, virtualisation, and availability at the edge have created many advantages for both users and service providers. The importance of securing this architecture has gained the attention of researchers working in this field. The works found in the literature employed a dataset to conduct their research, as opposed to employing a 5G mobile telecommunication testbed. Another fact to consider is that their respective research in traffic analysis and intrusion detection for 5G-MEC does not employ publicly available datasets collected from a mobile telecommunication 5G testbed. Currently, popular datasets include KDD Cup‘99 [45], NSL-KDD [46], CTU-13 [47] and UNSW NB-15 [48]. Of the above datasets, KDD Cup‘99 was employed for example in [49] and [50], the CTU dataset in [7], UNSW NB-15 in [51] and [52] and InSDN [53].

KDD Cup’99 was collected in 1999 from a U.S. Air Force LAN network with realistic network traffic. This labelled dataset albeit imbalanced contained a large number of entries. Although the dataset is quite popular amongst researchers, lack of variety in traffic, predefined attacks, repetitive data points, and being outdated questions its application for 5G research. NSL-KDD was created to improve the original 1999 dataset. Compared to the predecessor, this dataset contained fewer data points all of which were unique with no duplicated records. Despite the advantage of removing redundant records, being outdated, imbalanced, collected on a LAN network, lack of variety, and the limited set of pre-defined attacks make the acquired datasets questionable for use in 5G. The CTU dataset was created by the CTU University of Prague in 2011 on a campus network with a collection of labelled real network traffic with a diverse collection of attack traffic. However, CTU dataset is small in comparison to other datasets used for 5G research, imbalanced, collected on controlled conditions, and doesn’t represent the traffic of a mobile telecommunication network. Hence, the employment of the CTU dataset is questionable for 5G research. The UNSW NB-15 dataset was created in 2015 using the IXIA tool on three virtualised servers. The dataset contains a collection of labelled network traffic consisting of a diverse collection of attack traffic. However, the UNSW NB-15 dataset is imbalanced, collected using a simulation tool, and therefore can be construed as not suitable for 5G research. Finally, the recently collated InSDN dataset was collected at University College Dublin, emulating a campus network with SDN traffic.

Similar to the previous datasets, InSDN is imbalanced, collected using simulation tools, and doesn't represent mobile telecommunication traffic. All of the aforementioned datasets lack the following traffic which is vital for the function of mobile telecommunication. Traffic such as Quick UDP Internet Connections (QUIC), GPRS Tunnelling Protocol (GTP), S1 Application Protocol (S1AP), Stream Control Transmission Protocol (SCTP) and Simple Service Discovery Protocol (SSDP) are not represented in the above datasets. We, therefore, believe that it is time to develop a testbed to generate realistic datasets for 5G-based research. The dataset was generated adhering to the quality of data as outlined by the research presented in [54–57].

From a security perspective frameworks/applications for 5G have been presented in [58–60]. However, these studies do not include testbeds for their respective work but rather present frameworks based on literature for specific use cases (eMBB, URLLC, and mMTC). The inclusion of testbeds will, we believe provide a deeper understanding of the feasibility of the frameworks/applications. In our thesis, we present a mobile telecommunication testbed capable of transmitting, capturing, and processing various types of 5G mobile traffic.

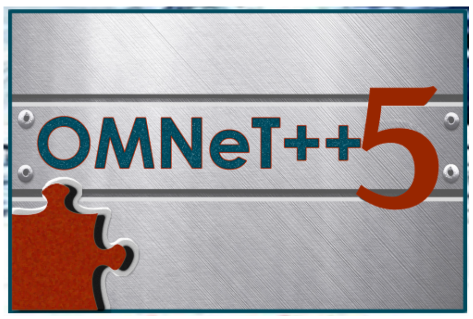
The heterogeneity, diversity and complexity of the applications due to complex use cases in 5G (and beyond), will we believe benefit in having a flexible parser that can carry out 'match+action' in parallel. Having the capability to re-configure, and to be both protocol and target (network devices i.e switches) independent, can aid in issues relating to heterogeneity, diversity and complexity for applications. For this research Programming Protocol Independent Packet Processing (P4), [16] serves as the prime candidate for inclusion in our testbed. In order to emphasise the importance of employing a realistic, relevant and pragmatic dataset for 5G-MEC security, we formulated a 5G-MEC testbed.

Simulation/emulation tool(s) such as NS3/Mininet has been used in [61–63] in their respective research in 5G, and do not involve data collected from a mobile telecommunication testbed. The above research raises the question as to the reliability of the study since the dataset or simulation/emulation does not involve traffic collected from a mobile telecommunication network.

Authors at [7, 49, 50, 64–72] researched towards resource management and security of 5G-MEC. The aforementioned studies did not utilise a testbed but merely used a publicly available dataset and an algorithm towards their respective study. This poses a question about the reliability of the study since the dataset does not represent the traffic found in a mobile telecommunication network. Traffic such as Quick UDP Internet Connections (QUIC) [73], GPRS Tunnelling Protocol [74], S1AP Protocol, Stream Control Transmission Protocol [75] and Simple Service Discovery Protocol (SSDP) were not represented in the datasets that the researchers have employed. Therefore, the employment of datasets towards research in 5G

(a) 5G-LENA simulator provided by NS3

(b) Mininet-WiFi Emulator



(c) OMNET++ Simulator

(d) Open Mobile Evolved Core (OMEC)

Fig. 2.3 Considered Simulators and Emulators for the development of a 5G mobile telecommunication testbed.

security creates a contradiction when their respective algorithms were faced with real traffic of a mobile network.

Frameworks for 5G mobile telecommunication have been reviewed by the authors at [76–79] without the employment of a testbed or an evaluation for their proposal. Security frameworks/applications for 5G have been presented in [58–60, 80] in their various studies. The studies given, do not include a testbed for their respective work but rather they have presented a framework-based literature for a specific use case. The literature referenced above, have presented their research without utilising a real mobile telecommunication testbed.

Fig. 2.4 Open Air Interface was employed for developing the 5G mobile telecommunication testbed

In order to develop a mobile telecommunication testbed following state-of-the-art deployments can be considered. They were: Open Mobile Evolved Core (OMEC) [81], Network Simulator 3 (NS3) 5G-LENA project [82], Omnet++ [83] and Mininet-wifi [84]. OMEC is an open-source deployment capable of launching a mobile telecommunication testbed with various customisations, offering opportunities to a varied set of experiments. High resource utilisation for the launch of OMEC, creates a bottleneck for deploying a 5G mobile telecommunication testbed. 5G-LENA is a recently developed open-source simulation tool capable of simulating various case studies and scenarios. However, a realistic 5G scenario or a case study, differs significantly from that of a simulated environment. Omnet++ is an open-source tool that offers modularity and visualisation for various case studies. However, the simulation nature and resource intensity question the suitability of Omnet++'s application. Finally, Mininet-wifi is an open-source tool created for educational and research purposes that possesses the capability to emulate custom topologies and case studies. Despite the popularity of this emulation tool, realism versus emulation differs significantly [85], contributing to the validity of the results collated. The high resource requirements for OMEC, the simulation nature of NS3 and Omnet++ and the missing LTE functions of Mininet-wifi show that these options are inappropriate for developing a 5G mobile telecommunication network. A better option we believe is the OpenAirInterface (OAI) which we have employed throughout this research.

OpenAirInterface [86] and [87] is an open-source development managed by the OpenAirInterface Software Alliance. OAI is a cost-effective tool with customisation options that support rapid prototyping that can be leveraged to launch a 5G mobile telecommunication testbed. However, OAI can not be optimised towards commercial use, but it enables researchers to test and evaluate 5G (and beyond) case studies and scenarios whilst maintaining compliance with the 3GPP standards for both core networks (CN) and radio access networks (RAN) [88].

OAI consists of the following emulating the CN: mobility management entity (MME), home subscriber server (HSS), and the control and user plane separation serving gateway (SPGW-C/U and PGW-C). OAI also deploys the components pertaining to the RAN, capable of connecting User Equipment to the network. To make the testbed realistic to reflect real-world applications, the following addenda were considered. Commercial Off the Shelf User Equipment (COTS-UE) was employed in connecting users to the network. This highlights a common user accessing medium via a smart UE. The users initiated the most popular types of network traffic in an attempt to highlight the user activity to the closest possible state. The network also consisted of connected but idle users to make the network more realistic. The network initialised services to users by employing UVT-Cloud nodes to demonstrate the popularity of the cloud services. A similar approach as above has also been employed by the authors in [87]. The employment of OAI and the addenda above will provide traffic patterns unique to a 5G mobile network, which can enhance the research of securing the 5G-MEC architecture.

2.4 Algorithms for the Detection Malicious Traffic

As the internet continues to evolve and the deployment of 5G and beyond expands, meeting strict low latency requirements will become increasingly important. In order to provide a service with low latency, high bandwidth and ultra-reliable communication to users, service providers will employ the concept of a Multi-Access Edge Computing (MEC) infrastructure. MEC and respective methods of deployment have been presented in [2] and as awareness and interest in 5G-MEC has grown within the industry and academia, research based on resource allocation, energy awareness and network slicing has also grown. Flexibility, scalability, virtualisation, and availability at the edge have created many advantages for both users and service providers. Given the potential and the advantages that the deployment provides, adversaries may target the platform to create service disruption [3]. Hence, the importance of securing the MEC infrastructure by applying a Network Intrusion Detection System (NIDS) has attracted attention from both academia and industry.

Despite the popularity of research in the field of NIDS technology, many applications remain based on signature-based methodologies for the detection of malicious traffic [4, 89–92]. Due to the increased volume of network data, lack of in-depth monitoring and granularity, as well as the diverse range of data types and protocols [4, 93] in a 5G mobile network, traditional signature-based NIDS systems may be less effective [94, 95]. With the complexity of non_malicious traffic increasing exponentially, the classification of traffic as malicious or non_malicious becomes a complex task [96]. DDoS attacks, spoofed attacks, port scan, SYN floods and other forms of malicious network traffic have become easier to launch with the availability of many open-source tools[97–99]. Therefore, malicious intent has become difficult to detect, especially when the attack is also spoofed [100]. Research by [101–106], suggests that the aforementioned attacks pose the most significant threat to MEC nodes due to low processing capabilities [103] and an attack may bypass a traditional NIDS [107]. An efficient method for the identification of malicious network traffic involves the utilisation of an efficient and effective signature-based detection algorithm [89, 108–110] that can label suspicious traffic [7]. A faster labelling [111, 112] of suspicious traffic can be extended by a cumulative and a confirmed detection [113] by an intelligent agent [7, 65, 114, 115], which has shown to be a more accurate approach than traditional NIDS [4] when signatures have been identified accurately [116]. Research [89, 93, 95, 117] suggest that this is the best source of detection. Similarly, an application of a behaviour-based NIDS will contribute towards a higher false positive rate [6] compared to a signature-based NIDS due to the high volume of data expected with 5G networks [114]. Signature-based NIDS can also be seen in applications where the devices have energy [118] and processing constraints [119]. When coupled with an intelligent agent (Machine Learning or Deep Learning), a signature-based NIDS can outperform a stand-alone signature-based NIDS as evidenced in [120, 121].

Interest in applying Machine Learning (ML) towards NIDS has grown exponentially [117] in recent years. ML techniques such as Naive Bayes (NB), Decision Trees (DT), Random Forrest (RF) and Support Vector Machines (SVM) have attracted the most attention for NIDS [89, 122, 123]. However, the inability to handle data with higher dimensions, the associated time overhead in training, the requirement for vast amounts of limited dimensional data and the labour-intensive process of identifying relevant data together with the heterogeneity of 5G data creates a bottleneck [4, 95] for the application of ML to NIDS problems.

Deep Learning (DL) has been promoted as a viable solution to the above-mentioned drawbacks of ML by extracting representations from network data relating to a DL modelling approach [5]. We note the most recent research into the NIDS problem based on DL techniques [89, 124–127]. Various studies have shown that DL approaches have achieved better results in comparison to ML techniques due to their advanced layer-based, feature learning mechanisms [128]. This benefits the analysis of network traffic with high complexity and dimensions [129], such as data generated through 5G mobile telecommunication and complicated applications (Massive Machine Type Communication and Ultra-Reliable Low Latency Communication) that occupy the network [130]. The capacity for learning from historical network data (both malicious and non-malicious) provides DL with the ability to reduce the complexity of network traffic in order to find correlations without human intervention [131, 124]. It is also noteworthy to mention that DL is capable of learning from a vast training dataset [132, 133] to build a detection model. After analysing the available DL techniques and as per the findings of Ding and Zhai [134], we employed Convolutional Neural Networks (CNN) due to its high performance [60, 135, 136] and accuracy [137–139].

Previous studies on the application of DL towards NIDS have been focused on testing the effectiveness of a desired feature selection or DL algorithm by employing a dataset/s [1, 60, 124, 130, 134, 140–151] that has been primarily collected on a network, which contradicts the traffic patterns and functions of a 5G network [152]. However, in order to test the hypothesis that the application of DL will be beneficial for NIDS on 5G networks, has to be tested by employing 5G data which is collected on a 5G mobile telecommunication testbed. Further to the employment of 5G data, the application of employing an intelligent agent such as a DL should be advanced from the traditional desk approach. Research in [130] also suggests that an algorithm trained and tested using a desk approach (off-line mode), may not function correctly in real-time.

Wu *et al.* [153] employed an algorithm employing all 122 features of the NSL-KDD Dataset into images which were later used towards training and testing a CNN. The mechanism employed by the researchers highlights the importance of utilising a Deep Learning algorithm instead of a Machine Learning algorithm where data with high dimension and complexity can be used. This research supports the use of CNN against other well-known classifiers, (Recurrent Neural Networks (RNN)). Fewer computations required for CNN and higher detection rates for DoS traffic were achieved in comparison to RNN, and also highlighted by the authors in their respective research. However, the research employed a dataset which was collected in 1999. The traffic patterns of the internet have changed significantly in comparison to that represented in the dataset.

Xiao *et al.* [154] presented research evaluating CNN, RNN and Deep Neural Networks (DNN), along with machine learning algorithms such as Logistic Regression, Decision Tree, Random Forest, Support Vector Machine, AdaBoost and Naive Bayes. In their research, they highlighted the performance on detection for the CNN algorithm against other well-known classifiers along with the time spent on training for CNN, RNN and DNN. The research highlighted and confirmed the high reputation of CNN against other classifiers which have been employed towards intrusion detection problems. Although the research provided promising results, the methodology employed utilised the KDD dataset. Authors at [155], employed the same dataset for their application of CNN-based IDS, evaluating it against other classifiers such as (DBN, SVM and Back Propagation Neural Network). Similar to the above approaches, authors at [156] also employed a CNN backed by the data from KDD and NSL-KDD datasets. The application of a dataset which was collected in the year 1999, questions the validity of results from the above research when the algorithm is placed in a real-time system for the detection of malicious 5G traffic.

Kim *et al.* [140] employed two datasets when evaluating the effectiveness of their CNN algorithm for intrusion detection problem. The authors have created images in both RGB and Grey-Scale. Upon image creation, the authors evaluated the effectiveness of their algorithm against different variations of convolutions and image classes. They employed two datasets for their research. Namely, they are KDD and CSE-CIC-IDS-2018 datasets. Authors identified that the application of RGB images produced effective and higher accuracy for detection compared to Grey-Scale images together with an application of 3 convolutional layers. A similar approach of employing 3 convolutional layers to increase the effectiveness of the detection algorithm can also be seen in the works of [135, 136, 138, 139, 155].

Research presented in [157] evaluated the effectiveness of various DL algorithms (CNN, Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU)) against the ISCX2012 dataset. In their research, the authors concluded that the application of CNN towards a binary traffic classification problem in an IDS produces higher results in comparison to other classifiers that they have employed in their work.

A hypothetical application of a real-time network intrusion detection system that conducts traffic classification at two stages was first introduced by the authors at [7]. This work presented a hypothetical signature detector following which confirmed traffic and suspicious network traffic was sent to an intelligent detection mechanism that employed a DL algorithm. The authors highlighted the importance of a two-stage detection mechanism for 5G and beyond to accommodate the higher volume and higher complexity of network traffic. However, the authors used the CTU dataset in order to test and evaluate the effectiveness of their DL algorithm as opposed to implementing the framework in a real-time testbed. Similarly, [158] presented a framework of detection using DL but the work is limited towards the application of CTU dataset as opposed to being applied to a real-time testbed. Work presented by [113], employed a simulator to collect GPS Tunnelling Protocol (GTP) packets and extrapolated features and detected DDoS attacks in real-time. The work presented highlighted the importance of a real-time network traffic classifier for 5G and beyond. However, the work is limited towards the detection of DDoS attacks. Also, the application of a testbed will provide more conclusive and reliable statistics as opposed to a simulator. Given the ability of DL to be trained with data which has higher dimensions and complexities ML will create a drawback towards the IDS problem when applied to 5G and beyond, which authors at [113] employed. Therefore, we hypothesise that the application of a DL algorithm that can conduct detection in real-time, in a 5G mobile telecommunication testbed will provide concrete results of its application and the generated accuracy.

2.5 Chapter Summary

The above section presents multiple areas of research available in the realm of 5G-MEC security. Research areas adjacent to the above progressed into the creation of new technologies and applications that can be integrated into solidifying contributions to the research questions. The next chapter in this thesis will present the evaluation undertaken for the first research question: How does programmability in both the control plan and data plane affect network performance?

Chapter 3

A Performance Evaluation for Software Defined Networks with P4

This chapter evaluates the performance of Software Defined Networking (SDN). SDN and Programming Protocol-independent Packet Processors (P4) have been attracting attention from both industry and academia as a means of achieving programmability in the network at different layers. The two emerging technologies have in the main, been researched individually in the literature, however, research evaluating the performance of SDN with P4 is limited. As the number of connected users continues to grow with the evolution of communication and with increasing complexity issues and heterogeneity relating to applications, the challenge for service providers to provide resilient connections with faster processing of packets increases. The application of SDN with its ease of management through centralised programmable control logic attracts attention from both academia and industry as a means of achieving better performance. This research was conducted to explore the hypothesis that combining programmability at both the control plane and data plane, namely *SDN+P4*, will provide a platform with faster packet processing. In this research, we design a system platform to investigate the network and applications performance in *SDN+P4*, in comparison to *SDN+Open vSwitch*. Topologies considered were the multi-path, grid and transit-stub network models with hosts placed at leaf nodes to emulate a small, an intermediate, and a core network. The mininet emulation results demonstrate that across all case studies, parallel processing of P4 has provided more queues for traffic to be processed with the utilisation of a flexible parser. With the evolution of the internet and the heterogeneity of applications, *SDN+P4* will, we believe, provide a more resilient stringent service to all use cases of 5G and beyond.

3.1 Introduction

Software Defined Networking (SDN) [11, 12] enables the separation of the control plane and the data forwarding plane, leading to enhanced performance through programmable network management. The acquired flexibility allows for virtualization and centralized control of the network, where routing tables are generated and updated centrally by a network controller. The protocol OpenFlow [13, 14] employs APIs to maintain forwarding rules for the switches establishing flow paths for data to travel in a network.

An OpenFlow forwarding rule consists of a ‘match’ and ‘action’. The ‘match’ involves the matching of a packet header fields such as source and destination fields and an ‘action’ involves the action to be performed such as forwarding or dropping a packet. This function is referred to as ‘match+action’ [15]. OpenFlow was first introduced with 12 ‘match’ header fields [16] and now stands at 44 header fields, as described in the OpenFlow specification 1.5.1 [14]. Anticipated to experience exponential growth in the foreseeable future, these developments respond to the evolving and heterogeneous nature of the internet. This could pose a potential issue, as the application of new header field extensions cannot be accomplished when employing a rigid, fixed parser during runtime, while the switches are actively processing packets.

An alternative to extending the OpenFlow specification is to employ a novel parser that can carry out ‘match+action’ in parallel allowing for programmability in the data forwarding plane without having to modify the OpenFlow specification. A parser of this nature can be updated, extended, and modified seamlessly without disrupting packet processing and forwarding, thereby enabling the integration of heterogeneous applications. A protocol that can be employed to carry out ‘match+action’ in parallel is the Programming Protocol Independent Packet Processing (P4) [16]. Parallel processing can be achieved in P4 by employing one of the following.

1. Multiple Processing Cores: Some network devices have multiple CPU cores or processing units. P4 programs can be designed to distribute packet processing tasks across these cores to achieve parallelism. For example, one core might handle packet parsing, while another core handles packet forwarding.

2. Parallel Pipelines: P4 allows the definition of multiple packet processing stages within a pipeline. These stages can be designed to operate in parallel, with each stage processing packets independently. For instance, one stage might perform access control checks while another stage performs packet routing.

3. Hardware Offload: In some cases, P4 programs can be used to offload certain packet processing tasks to specialized hardware accelerators or programmable ASICs. These hardware components can operate in parallel with the CPU, further increasing packet processing efficiency.

4. Load Balancing: P4 can be used to implement load balancing mechanisms, where incoming packets are distributed across multiple processing units or network paths, enabling parallel processing of packets.

P4 possesses three primary properties: re-configurability, protocol independence, and target independence. These features empower the network administrator to define the functions and capabilities of a switch, rather than being constrained by a vendor's specifications [17]. By utilizing a common open interface, administrators can leverage P4 to program a flexible parser for matching new header fields. This stands in contrast to working with a fixed parser in OpenFlow, where processing new header fields would necessitate a specification update.

The extensive research carried out in [18, 19] indicate that the majority of future network traffic is expected to be the result of using hand-held smart devices. It is accounted that the increased traffic due to hand-held devices has resulted an approximate 82% of all consumer Internet traffic. The escalating load of end-user IP traffic poses a potential risk of network congestion, which could lead to a diminished Quality of Service (QoS) in terms of factors such as jitter, packet loss, and throughput, as noted in [21]. Service providers are actively working to uphold an acceptable level of service amid these challenges. It is noteworthy to highlight a challenge outlined in [22], indicating that a significant portion of the delay in data communication arises in the core of the network. This is attributed to the exponential growth in periodic updates [23] employed for maintaining the network state.

Control and data plane programmable functions motivate us to investigate how this emerging approach can improve the performance of a network by processing packets faster. We believe it is essential to examine whether and how a solution incorporating programmability in both the control plane and forwarding plane can offer an enhanced core network capable of meeting stringent performance requirements and providing flexibility as network traffic continues to increase.

We hypothesise that a combination of data plane and control plane programmability (*SDN+P4*) will elevate the performance of the network, to accommodate stringent performance requirements for future applications in comparison to 'Open vSwitch (OvS) [44] coupled with control plane programmability' (*SDN+OvS*). In this chapter, we explore *SDN+OvS* and *SDN+P4* environments for different topologies comparing the performance of each. Various types of traffic were transmitted to test the performance of the *SDN+OvS* and *SDN+P4*.

3.2 System Platforms

To explore the research questions, a network emulation was built on Ubuntu 18.04LTS running on VMWare with a Core i7 CPU with 3.40GHz together with an Open Networking Operating System (ONOS) [159] version 2.0.0. Mininet [84] was utilised to emulate the networks, as shown in Fig.3.1.

3.2.1 SDN Platform

ONOS is an open-source SDN network operating system built to provide a platform for service provider networks [159]. ONOS facilitates the control plane for network components such as switches, links or software [160]. Being highly reliable, resilient, and scalable makes ONOS, we believe the best open-source option [161, 162] available for building and catering for SDN networks. A real-time GUI provides a global view of the network that aids administrators to monitor and manage resources accordingly. ONOS provides two APIs: the southbound API interacts with devices and the northbound API offers services to applications. The controller inside ONOS can push a switch configuration towards a forwarding device. We chose ONOS as a platform in this research for the above features and its ability to maintain a network's state without compromising performance.

3.2.2 Mininet

Mininet [84], developed by Stanford University, is an open-source, lightweight and easy-to-deploy network emulator that provides a programmable interface to launch and initialise networks either in wireless or wired mode. Mininet has the ability to initialise a large network with multiple hosts and switches on a physical host. Mininet supports the emulation of OpenFlow enabled switches such as OvS for this research. OvS coupled with an ONOS controller was used to emulate the *SDN+OvS* environment for this research.

3.2.3 P4 Switch

Upon selecting ONOS for the SDN platform, we chose the bmv2 switch [163] as our P4 software switch. A bmv2 switch can configure a custom parser with ingress and egress pipelines working in parallel to perform match+action. In bmv2, parsers are implemented

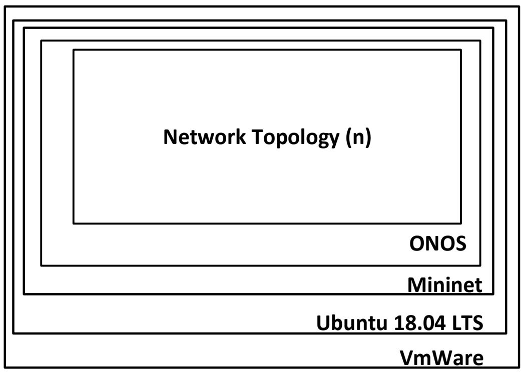


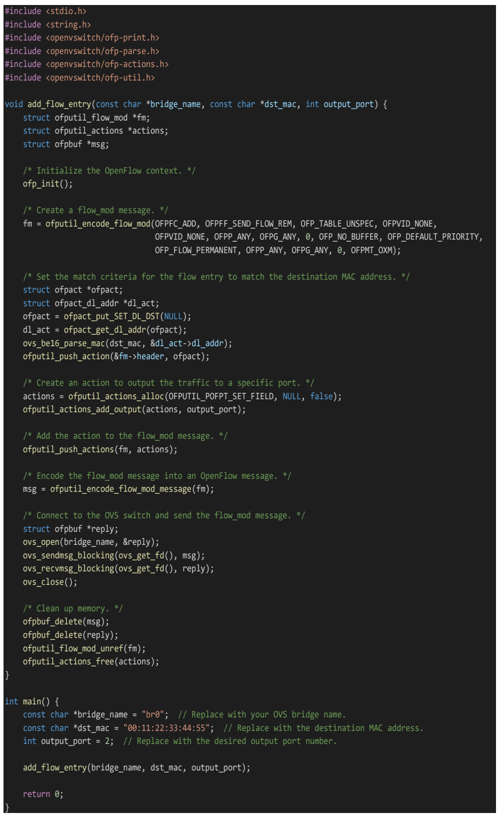
Fig. 3.1 Test bed used for this research.

prior to the match+action stages. As we employed the standard bmv2 to emulate a P4 software switch, the parsers are implemented in this manner by default. The program utilised is independent of the target switch design and can be employed to represent different switch designs should the need arise. The P4 software switch, bmv2 was instantiated with parallel processing and without using specific metadata or specific port forwarding to specific traffic (dynamic features of P4) to ensure fairness between the OvS switch and the bmv2 switch. The bmv2 switch coupled with the ONOS controller was employed to emulate the *SDN+P4* environment in this research.

SDN+P4 will enable programmability in both the control and data forwarding plane. Unlike the previous instance (*SDN+OvS*), *SDN+P4* has the capability to modify the switch configuration such as a flexible parser, 'match+action' which can operate in parallel or series or include metadata and buffer, using a JavaScript Open Notation (JSON) obtained from a P4 program [16]. By utilising a P4 switch, a fixed parser in OvS has been replaced with a flexible parser, which can process packets more efficiently and effectively. Fig.3.2a highlight a code written in C to implement an Open VSwitch that can conduct match+action based on MAC address. Fig. 3.2b, depicts the same functionality that has been implemented using P4. In this code, the network packet parameters can be configured to achieve flexibility and programmability in the data forwarding plane, which can not be achieved in OvS.

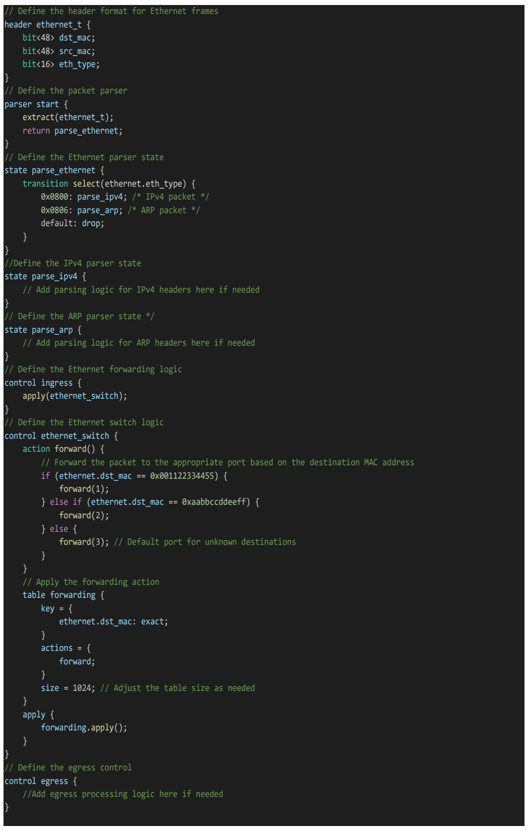
3.3 Experimental Design

Experiments designed for answering the research question "How does programmability in both the control plane and data plane affect network performance?", have been conducted as per Table 3.1. In this chapter, the computational complexity or capacity to implement bmv2 or OvS were not collected, as this was outside the set research goals.



```c
#include <stdio.h>
#include <string.h>
#include <openvswitch/ofp-print.h>
#include <openvswitch/ofp-parse.h>
#include <openvswitch/ofp-actions.h>
#include <openvswitch/ofp-util.h>

void add_flow_entry(const char *bridge_name, const char *dst_mac, int output_port) {
    struct ofputil_flow_mod *fm;
    struct ofputil_actions *actions;
    struct ofpbuf *msg;

    /* Initialize the OpenFlow context. */
    ofp_init();

    /* Create a flow_mod message. */
    fm = ofputil_encode_flow_mod(OFPFC_ADD, OFPFF_SEND_FLOW_REM, OFP_TABLE_UNSPEC, OFPVID_NONE,
                                 OFPVID_NONE, OFPP_ANY, OFPG_ANY, 0, OFP_NO_BUFFER, OFP_DEFAULT_PRIORITY,
                                 OFP_FLOW_PERMANENT, OFPP_ANY, OFPG_ANY, 0, OFPMT_OXM);

    /* Set the match criteria for the flow entry to match the destination MAC address. */
    struct ofpact *ofpact;
    struct ofpact_dl_addr *dl_act;
    ofpact = ofpact_put_SET_DL_DST(NULL);
    dl_act = ofpact_get_dl_addr(ofpact);
    ovs_be16_parse_mac(dst_mac, &dl_act->dl_addr);
    ofputil_push_action(&fm->header, ofpact);

    /* Create an action to output the traffic to a specific port. */
    actions = ofputil_actions_alloc(OFPUTIL_POFPT_SET_FIELD, NULL, false);
    ofputil_actions_add_output(actions, output_port);

    /* Add the action to the flow_mod message. */
    ofputil_push_actions(fm, actions);

    /* Encode the flow_mod message into an OpenFlow message. */
    msg = ofputil_encode_flow_mod_message(fm);

    /* Connect to the OVS switch and send the flow_mod message. */
    struct ofpbuf *reply;
    ovs_open(bridge_name, &reply);
    ovs_sendmsg_blocking(ovs_get_fd(), msg);
    ovs_recvmsg_blocking(ovs_get_fd(), reply);
    ovs_close();

    /* Clean up memory. */
    ofpbuf_delete(msg);
    ofpbuf_delete(reply);
    ofputil_flow_mod_unref(fm);
    ofputil_actions_free(actions);
}

int main() {
    const char *bridge_name = "br0";  // Replace with your OVS bridge name.
    const char *dst_mac = "00:11:22:33:44:55";  // Replace with the destination MAC address.
    int output_port = 2;  // Replace with the desired output port number.

    add_flow_entry(bridge_name, dst_mac, output_port);

    return 0;
}
```

(a) A Sample Open vSwitch code



```
// Define the header format for Ethernet frames
header ethernet_t {
    bit<48> dst_mac;
    bit<48> src_mac;
    bit<16> eth_type;
}
// Define the packet parser
parser start {
    extract(ethernet_t);
    return parse_ethernet;
}
// Define the Ethernet parser state
state parse_ethernet {
    transition select(ethernet.eth_type) {
        0x0800: parse_ipv4; /* IPv4 packet */
        0x0806: parse_arp; /* ARP packet */
        default: drop;
    }
}
//Define the IPv4 parser state
state parse_ipv4 {
    // Add parsing logic for IPv4 headers here if needed
}
// Define the ARP parser state */
state parse_arp {
    // Add parsing logic for ARP headers here if needed
}
// Define the Ethernet forwarding logic
control ingress {
    apply(ethernet_switch);
}
// Define the Ethernet switch logic
control ethernet_switch {
    action forward() {
        // Forward the packet to the appropriate port based on the destination MAC address
        if (ethernet.dst_mac == 0x001122334455) {
            forward(1);
        } else if (ethernet.dst_mac == 0xaabbccddeeff) {
            forward(2);
        } else {
            forward(3); // Default port for unknown destinations
        }
    }
    // Apply the forwarding action
    table forwarding {
        key = {
            ethernet.dst_mac: exact;
        }
        actions = {
            forward;
        }
        size = 1024; // Adjust the table size as needed
    }
    apply {
        forwarding.apply();
    }
}
// Define the egress control
control egress {
    //Add egress processing logic here if needed
}
```

(b) A Sample P4 code

Fig. 3.2 Sample code of OvS and P4

3.3.1 Network Topologies

Fig.3.3 illustrates the three topologies that we employed. Topology I is a multi-path topology that extends the spine-leaf [164] topology used by data centres.Multi-path topology was employed to test the effectiveness of *SDN+OvS* and *SDN+P4* to a popular topology employed to simulate a data centre network. Gradual increment of nodes from [165] and ease of programming also contributed towards the rationality for the multi-path topology. Topology II was employed to emulate an environment with a simple-grid topology and a similar topology has been presented in [41, 166, 167]. Further extension of the multi-path topology with more nodes to the network and the popularity of similar topologies being employed for SDN research were the rationale towards employing the simple grid topology. Topology III emulates the transit-stub network model also known as the Internet topology initially

Experiments
RQ1: Evaluation of Control Plane and Data Plane
1. Emulation of Control Plane and Data Plane Programmability
2. Emulation of Different Topologies
3. Emulation of Various Network Traffic Types
4. Analysis of Key Performance Indicators

Table 3.1 Experiments conducted in this Chapter

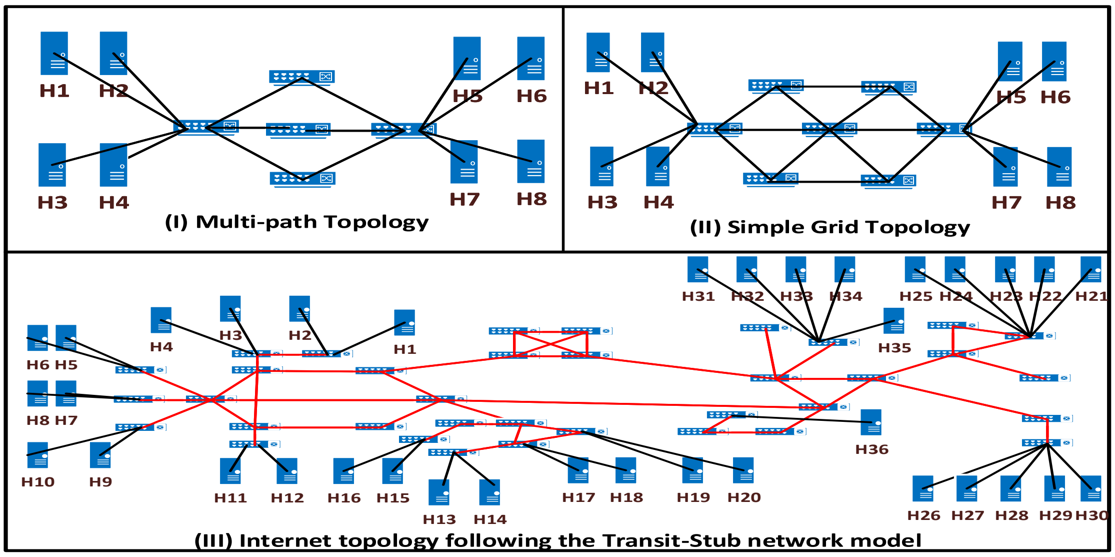


Fig. 3.3 Topologies employed for this research,
(I) multi-path topology (II) grid topology and a
(III) Transit stub network model. (H denotes Host)

presented in [34] and in subsequent works [33, 168–174]. Extensive research conducted towards an SDN based internet, and the popularity of similar topologies being employed for such research were the rationale towards employing the simple grid topology.

Traffic	Topology I and Topology II	Topology III
	Client → Server	Client → Server
ICMP	H1 → H5, H2 → H6, H3 → H7, H4 → H8	H1 → H21, H5 → H22, H9 → H23, H13 → H24, H17 → H25
TCP	H1 → H5, H2 → H6, H3 → H7, H4 → H8	H2 → H26, H6 → H27, H10 → H28, H14 → H29, H18 → H30
UDP	H1 → H5, H2 → H6, H3 → H7, H4 → H8	H3 → H31, H7 → H32, H11 → H33, H15 → H34, H19 → H35
CDN	H1 → H8, H2 → H8, H3 → H8, H4 → H8	H4 → H36, H8 → H36, H12 → H36, H16 → H36, H20 → H36

Table 3.2 Configuration of traffic. Hosts are shown in Fig.3.3.

3.3.2 Traffic Design

In order to mimic the Internet as closely as possible, traffic types such as Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP) [175], User Datagram Protocol (UDP) [176] and Content Delivery Network (CDN) were chosen. The traffic types were selected based on research presented in [177], where the results focused on popular downloaded traffic. ICMP traffic is widely used as a measurement tool for troubleshooting, control and error messaging services. TCP can be considered as one of the main protocols of the IP suite. TCP is considered as one of the most trustworthy and reliable protocols utilised by applications on the internet. UDP is widely used to send messages with a minimum protocol mechanism. Based on the research presented in [178] and the popularity of UDP applications, UDP traffic was included. Finally with the popularity and demand of the CDN as highlighted in [18, 179] a case study emulating the same has been included. The above distinct traffic types were examined both on an individual basis and mixed basis to test and evaluate the performance of the network for both *SDN+OvS* and *SDN+P4*.

The involved experiments were conducted in two stages: Tier 1, in which only one type of traffic at a time was run over the network, ICMP, TCP, UDP or CDN, each forming a single case study and Tier 2, in which all four types of traffic were run simultaneously over topologies II and III.

3.3.3 Tier-I - Single Type of Traffic Run

Table 3.2 shows the configuration of simple types of traffic run in this Tier.

Case Study 1 - ICMP

A custom Python script was utilised to send ICMP traffic between the designated hosts simultaneously to emulate an unpredictable traffic load in the network. ICMP traffic was generated using the ping command. A total of 1000 ICMP sequenced packets were saved for calculations for the topologies I and II whilst 2000 sequenced packets for topology III with a packet size of 1500 bytes and a data rate of 12 Kbps. Data was saved for both *SDN+OvS* and *SDN+P4* in each of the three topologies.The default ping command encompasses the aforementioned packet size and the data rate. Hence the rationality in choosing these configurations. The number of ICMP packets transferred for topologies I and II was chosen for ease of calculation. Due to the size of the network, the number of ICMP packets saved was doubled for topology III.

Case Study 2 - TCP Traffic

In order to establish a connection with the server and the client and to transmit data between the client and the server, iPerf[180] v2.0 was used. iPerf's capabilities and functions were thoroughly examined and presented in [177]. Given the stability of v2.0 over v3.0 in iPerf, the former was selected for the experiments. For topologies, I and II, data with a file size limitation of 1GB encapsulated, as TCP was chosen. For topology III, a data burst for 1200s was chosen to identify which network abstraction was able to process the largest amount of data. For ease of emulation and calculation, a download with the file size of 1GB was chosen for Topologies I and II. The 1GB file size served as an example to a scenario where a user intends to download a file of a fixed size from an File Transfer Protocol (FTP) server. For a scenario where the user is engaged in a TCP transmission with a time constraint (i.e., Secure Shell (SSH) or Telnet), a time limitation of 1200s was employed for the topology III. This was also applied with respect to the size of the network and for extending the download time and the amount of data.

Case Study 3 - UDP Traffic

For this case study, iPerf was utilised to stream UDP traffic. Similar to case study 2, the use of 1GB traffic for topologies I and II with a data burst of 1200s for topology III. The iPerf traffic was initiated with a bound of 12MBps between the client and the server for both UDP due to resource limitations in the system platform. Similar to the TCP case study, a download with the file size of 1GB was chosen for Topologies I and II. This case study setting was designed with the intent of emulating a user downloading multimedia content, which will be bound by a file size. For topology III, a UDP download was conducted for

1200s in the aim of emulating a user engaged in online gaming or video conference. Time constraint of 1200s was also applied with respect to the size of the network and for extending the download time and the amount of data. Measurements such as throughput, delay jitter and transmission completion time were collected for both *SDN+OvS* and *SDN+P4* for each of the three topologies.

Case Study 4 - Content Delivery Network

An emulation was designed to explore video traffic with live stream, using VLC-player [181] to send videos with the quality of 1080p and H.264 compression due to the demand and the popularity of high-quality video streaming. For topology, I and II, a video of 600s long was chosen and for topology III a video of similar quality was chosen with 1200s. The length of the two video files (600s and 1200s) was used for the ease of emulation and calculation purposes. As the topology III was larger in size as opposed to Topologies I and II, a larger video file was employed. Topology III emulated a higher number of hosts and switches involved. In order to establish an emulation reinforcing a live video stream, a Python script was utilised that determined transfers between multiple clients with a server.

3.3.4 Tier-II - Multiple Types of Traffic Running Simultaneously

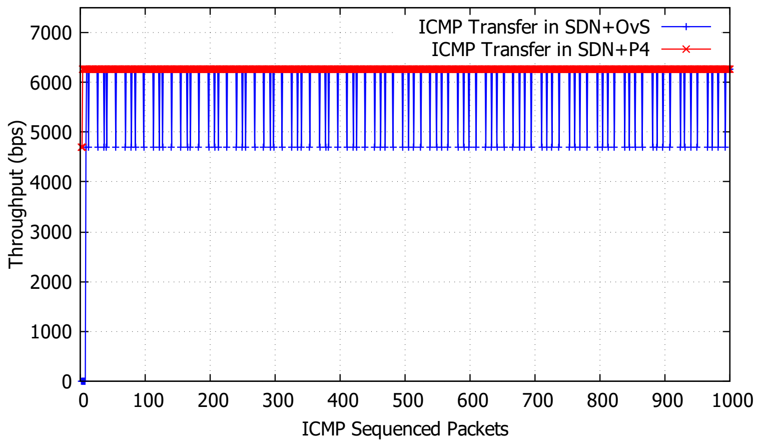
Networks will carry different types of traffic executing simultaneously. Tier II involves two case studies. The first look at, Simultaneous Traffic in Topology II and the second looks at Simultaneous Traffic in Topology III. Simultaneous Traffic refers to the initiation of all traffic shown in Table 3.2.

Case Study 5 - Simultaneous Traffic in Topology II

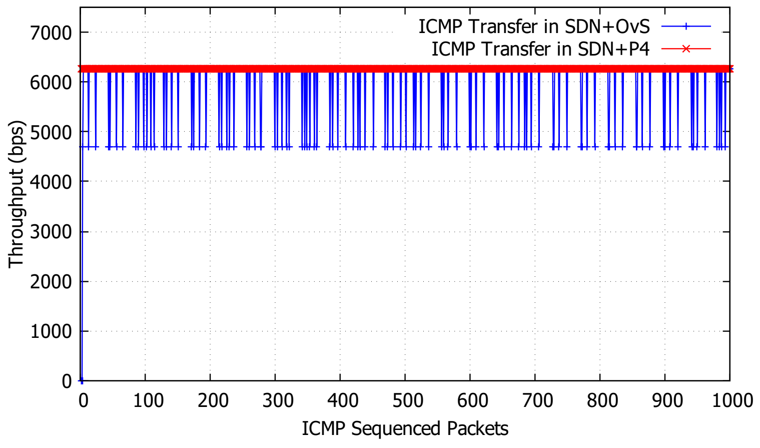
A custom Python script was utilised all traffic in Table 3.2 at the same time in an attempt to saturate links with applications being launched at the same time. Client and server configurations remain the same for this case study, as they were for individual case studies to ease of evaluation since the variance in performance can be easily observed between the individual execution and simultaneous execution. The distinction of traffic types and their usage remains the same with the addition of VoIP traffic. VoIP traffic was not presented individually since the results between *SDN+OvS* and *SDN+P4* had less significance due to a smaller volume of traffic involved in the communication.

Case Study 6 - Simultaneous Traffic in Topology III

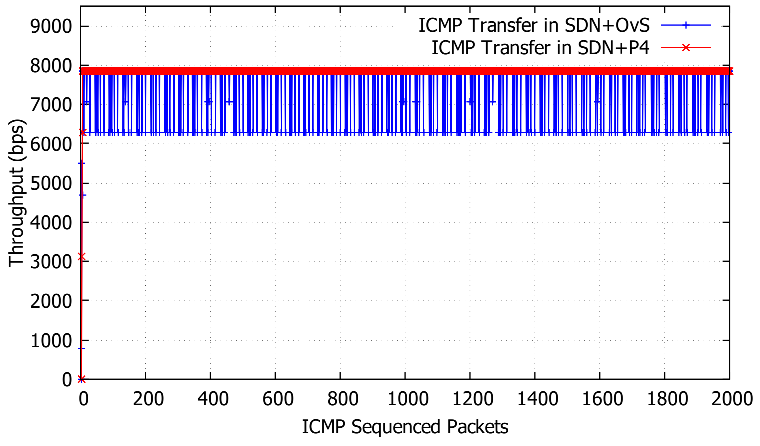
This case study launch all traffic simultaneously according to Table 3.2 in the Internet topology (Topology III). This approach aids in closely mimicking the Internet in which multiple clients and servers are sending and receiving traffic in the network. Since the network is under stress due to the amount of traffic and applications, the collected data



(a) ICMP transfer of 1000 packets in simple multi-path topology

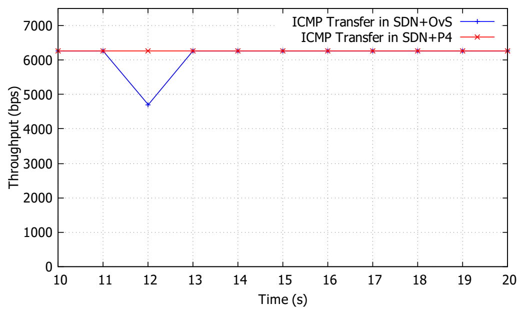


(b) ICMP transfer of 1000 packets in simple grid topology

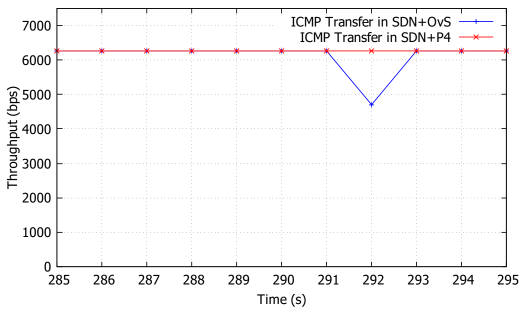


(c) ICMP transfer of 2000 packets in transit-stub network model

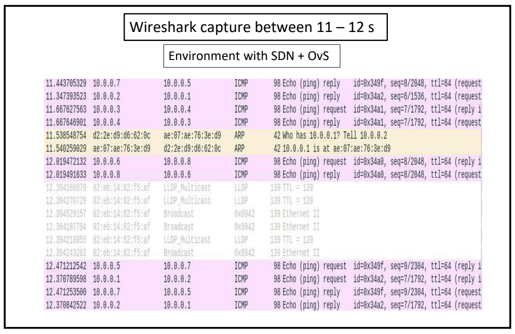
Fig. 3.4 Throughput of ICMP transfer in case study 1



(a) ICMP throughput at 10-20s in topology I

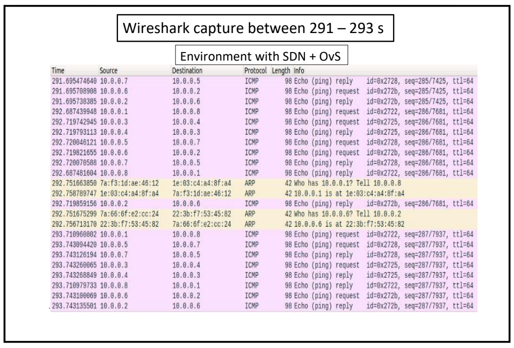


(b) ICMP throughput at 285-295s in topology I


Wireshark capture between 11 – 12 s
Environment with SDN + OvS

```
11.443705329  10.0.0.7           10.0.0.5       ICMP    98 Echo (ping) reply    id=0x349f, seq=8/2048, ttl=64 (request
11.347393523  10.0.0.2           10.0.0.1       ICMP    98 Echo (ping) reply    id=0x34a2, seq=6/1536, ttl=64 (request
11.667627563  10.0.0.3           10.0.0.4       ICMP    98 Echo (ping) request  id=0x34a1, seq=7/1792, ttl=64 (reply i
11.667646901  10.0.0.4           10.0.0.3       ICMP    98 Echo (ping) reply    id=0x34a1, seq=7/1792, ttl=64 (request
11.538548754  d2:2e:d9:d6:62:0c  ae:07:ae:76:3e:d9  ARP  42 Who has 10.0.0.1? Tell 10.0.0.2
11.540259029  ae:07:ae:76:3e:d9  d2:2e:d9:d6:62:0c  ARP  42 10.0.0.1 is at ae:07:ae:76:3e:d9
12.019472132  10.0.0.6           10.0.0.8       ICMP    98 Echo (ping) request  id=0x34a0, seq=8/2048, ttl=64 (reply i
12.019491633  10.0.0.8           10.0.0.6       ICMP    98 Echo (ping) reply    id=0x34a0, seq=8/2048, ttl=64 (request
12.394160070  02:eb:14:82:f5:af  LLDP_Multicast LLDP   139 TTL = 120
12.394270729  02:eb:14:82:f5:af  LLDP_Multicast LLDP   139 TTL = 120
12.394529157  02:eb:14:82:f5:af  Broadcast      0x8942 139 Ethernet II
12.394187704  02:eb:14:82:f5:af  Broadcast      0x8942 139 Ethernet II
12.394216055  02:eb:14:82:f5:af  LLDP_Multicast LLDP   139 TTL = 120
12.394243263  02:eb:14:82:f5:af  Broadcast      0x8942 139 Ethernet II
12.471212542  10.0.0.5           10.0.0.7       ICMP    98 Echo (ping) request  id=0x349f, seq=9/2304, ttl=64 (reply i
12.370789598  10.0.0.1           10.0.0.2       ICMP    98 Echo (ping) request  id=0x34a2, seq=7/1792, ttl=64 (reply i
12.471253500  10.0.0.7           10.0.0.5       ICMP    98 Echo (ping) reply    id=0x349f, seq=9/2304, ttl=64 (request
12.370842522  10.0.0.2           10.0.0.1       ICMP    98 Echo (ping) reply    id=0x34a2, seq=7/1792, ttl=64 (request
```

(c) Wireshark capture of ARP and LLDP messages at 11-12s in topology I


Wireshark capture between 291 – 293 s
Environment with SDN + OvS

```
Time          Source             Destination        Protocol Length Info
291.695474640 10.0.0.7           10.0.0.5           ICMP     98 Echo (ping) reply    id=0x2728, seq=285/7425, ttl=64
291.695708908 10.0.0.6           10.0.0.2           ICMP     98 Echo (ping) request  id=0x272b, seq=285/7425, ttl=64
291.695738385 10.0.0.2           10.0.0.6           ICMP     98 Echo (ping) reply    id=0x272b, seq=285/7425, ttl=64
292.687439948 10.0.0.1           10.0.0.8           ICMP     98 Echo (ping) request  id=0x2722, seq=286/7681, ttl=64
292.719742945 10.0.0.3           10.0.0.4           ICMP     98 Echo (ping) request  id=0x2725, seq=286/7681, ttl=64
292.719793113 10.0.0.4           10.0.0.3           ICMP     98 Echo (ping) reply    id=0x2725, seq=286/7681, ttl=64
292.720046121 10.0.0.5           10.0.0.7           ICMP     98 Echo (ping) request  id=0x2728, seq=286/7681, ttl=64
292.719821655 10.0.0.6           10.0.0.2           ICMP     98 Echo (ping) request  id=0x272b, seq=286/7681, ttl=64
292.720070588 10.0.0.7           10.0.0.5           ICMP     98 Echo (ping) reply    id=0x2728, seq=286/7681, ttl=64
292.687481604 10.0.0.8           10.0.0.1           ICMP     98 Echo (ping) reply    id=0x2722, seq=286/7681, ttl=64
292.751663850 7a:f3:1d:ae:46:12  1e:03:c4:a4:8f:a4  ARP      42 Who has 10.0.0.1? Tell 10.0.0.8
292.758789747 1e:03:c4:a4:8f:a4  7a:f3:1d:ae:46:12  ARP      42 10.0.0.1 is at 1e:03:c4:a4:8f:a4
292.719859156 10.0.0.2           10.0.0.6           ICMP     98 Echo (ping) reply    id=0x272b, seq=286/7681, ttl=64
292.751675299 7a:66:6f:e2:cc:24  22:3b:f7:53:45:82  ARP      42 Who has 10.0.0.6? Tell 10.0.0.2
292.756713170 22:3b:f7:53:45:82  7a:66:6f:e2:cc:24  ARP      42 10.0.0.6 is at 22:3b:f7:53:45:82
293.710960802 10.0.0.1           10.0.0.8           ICMP     98 Echo (ping) request  id=0x2722, seq=287/7937, ttl=64
293.743094420 10.0.0.5           10.0.0.7           ICMP     98 Echo (ping) request  id=0x2728, seq=287/7937, ttl=64
293.743126194 10.0.0.7           10.0.0.5           ICMP     98 Echo (ping) reply    id=0x2728, seq=287/7937, ttl=64
293.743260065 10.0.0.3           10.0.0.4           ICMP     98 Echo (ping) request  id=0x2725, seq=287/7937, ttl=64
293.743268849 10.0.0.4           10.0.0.3           ICMP     98 Echo (ping) reply    id=0x2725, seq=287/7937, ttl=64
293.710979733 10.0.0.8           10.0.0.1           ICMP     98 Echo (ping) reply    id=0x2722, seq=287/7937, ttl=64
293.743100069 10.0.0.6           10.0.0.2           ICMP     98 Echo (ping) request  id=0x272b, seq=287/7937, ttl=64
293.743135501 10.0.0.2           10.0.0.6           ICMP     98 Echo (ping) reply    id=0x272b, seq=287/7937, ttl=64
```

(d) Wireshark capture of ARP and LLDP messages at 291-293s in topology I

Fig. 3.5 ICMP data capture at 10-20s and 285-295s in topology I

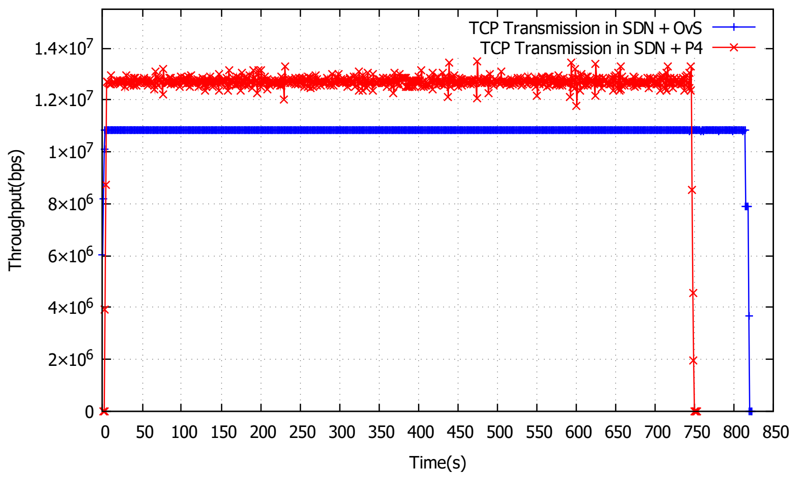
provides valuable results. Delay, delay jitter, packet loss, bps and throughput are amongst the key statistics recorded at client-side.

3.4 Results and Analysis of Tier-I Single Type of Traffic Run

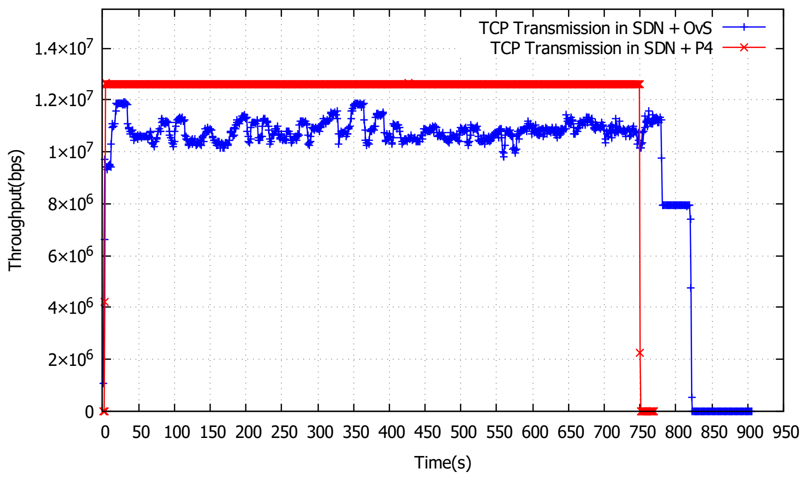
This section will present and analyse the results of the experiments for Tier-I, a single type of traffic run. Data of both *SDN+OvS* and *SDN+P4* will be presented under each case study followed by their analysis.

3.4.1 Case Study 1 - ICMP

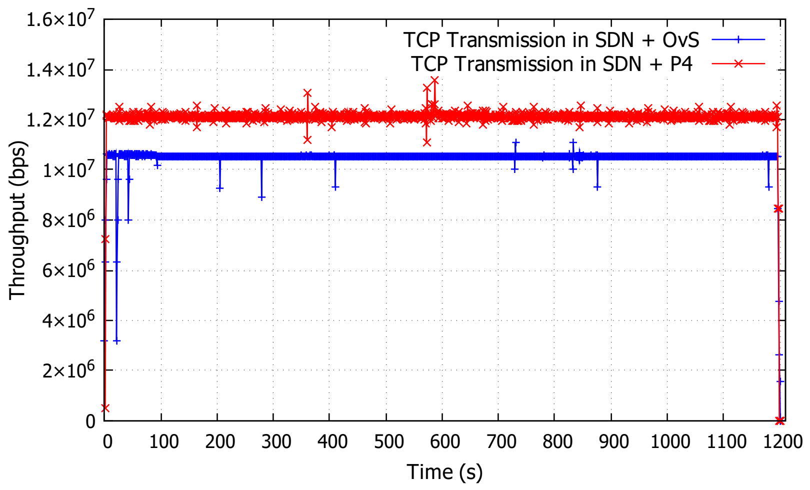
The ICMP traffic for the configuration given in Table 3.2 commenced at the same time. At the client end, Wireshark collected 1000 ICMP packets. Following the completion of the transfer, the average throughput for each connection was recorded.



(a) TCP transfer of 1GB data using iPerf in topology I



(b) TCP transfer of 1GB data using iPerf in topology II



(c) TCP transfer for 1200s using iPerf in topology III

Fig. 3.6 Throughput of TCP data transfer in case study 2

Fig.3.4 compares the ICMP throughput for *SDN+OvS* networking and *SDN+P4* networking, over the three topologies in Fig.3.1. For topology I (multi-path) and topology II (grid), the *SDN+P4* networking has maintained a steady stream of ICMP traffic with a constant throughput of approximately 6200bps until the transmission terminates (see the red line in Figs.3.4a and 3.4b). The *SDN+OvS* emulation recorded recurring drops in throughput from approximately 6200bps to 4800bps (see the blue lines in Figs.3.4a and 3.4b). The same pattern in Fig.3.4c, the transit-stub network model, with values of approximately 7800bps for *SDN+P4* and recurring drops in throughout to approximately 6200bps for *SDN+OvS*. However, in Fig.3.4c, the throughput have registered an aperiodic drop for *SDN+OvS* as depicted by blue lines. This can be due to one of the two following factors, processing requirements or congestion. When faced with processing requirements or congestion, SDN networks tend to face performance degradation and cause packet loss. The same has also been observed by [42].

We investigated the throughput drop in *SDN+OvS* experiments. Fig.3.5 shows the throughput drops at 12s and 292s (Figs.3.5 (a) and 3.5(b)) and the Wireshark captures Figs.3.5 (c) and 3.5 (d)) at those time for topology I. Out of the two controller states, reactive and proactive, the reactive state was chosen to avoid miss-configurations during the installation of FlowRules using the Reactive Forwarding application in the ONOS controller. Given that the controller was in a reactive state, PACKET_IN and PACKET_OUT control messages played a crucial role in maintaining ARP/IP address mapping information [182]. The drop in throughput for *SDN+OvS* resulted from the ARP and LLDP packets (observed in Fig.3.5c and Fig.3.5d) being routed towards the slow path of the OvS, detouring from the caching layer. A detour occurs when the OvS sends a PACKET_IN message to the SDN controller to resolve the headers. Until a response arrives as a form of PACKET_OUT from the SDN controller, the packet will remain in the switch buffer. Deviating from the caching layer or fast path can create an overhead [183] in the network which leads to a deterioration in performance. Research in [23, 184, 185] established that ARP messages can create a bottleneck in the network due to high volume and frequency [186–188]. Therefore deviating from a fast path in the OvS architecture creates a substantial overhead in the network. Processing requirements from the CPU towards the ARP/LLDP packets can also affect the processing of application centric packets ultimately resulting in a poor quality of service. Since each emulated host represents a networked host with the OS as Ubuntu 18.04, the default ARP update time is 60s. Hence a periodic ARP update will be generated automatically. The results in Fig 3.4 and 3.5 demonstrate how the network performance has been affected in the SDN+OvS architecture through periodic ARP updates.

In the *SDN+P4* experiments, the P4 switch processed headers in parallel, thus the LLDP and ARP packets did not interfere with ICMP packets in the pipeline. Hence the *SDN+P4* environment did not experience a throughput drop (red line in Fig.3.4). Due to parallel processing for headers and packets, the bmv2 switch was able to process packets faster, reducing delay in the core network. Given the periodic nature of the ARP packets (60s by default), the exponential growth associated with a higher number of connected devices and ARP storms caused by network outages a bottleneck may if the packets are not processed faster. Hence utilising *SDN+P4* will aid in achieving some stringent requirements for future applications given its ability to process packets faster for both application centric and network centric traffic.

3.4.2 Case Study 2 - TCP

TCP traffic was generated via a file download of 1GB using iPerf for topologies I and II. For topology III a transmission of TCP for 1200s was employed. See Fig. 3.1 and Table 3.2 for respective topologies and traffic configuration. The results for the experiments are shown in Fig.3.6 with a summary of the results given in Table 3.3.

In *SDN+P4* (red line) TCP has achieved a higher and more constant throughput in comparison to *SDN+OvS* (blue line). In Figs.3.6a and 3.6b, the fixed sized TCP download completed earlier for SDN+P4 in approximately 750s as opposed to approximately 820s for *SDN+OvS*. In Fig.3.6b, blue line experienced fluctuations of throughput. TCP traffic, by design, requires acknowledgments. As the data packets and the acknowledgments traverse in the same route (as observed by accessing the ONOS GUI), the throughput fluctuated due to saturation of links. It is also noteworthy that for both *SDN+P4* and *SDN+OvS*, TCP traffic traversed in the same route for topology II. However, the *SDN+P4* didn't record a fluctuation of throughput due to faster processing capability. For topology I, (Fig.3.6a) a fluctuation of throughput was not recorded in the *SDN+OvS* due to the small size of the network and fewer nodes with processing requirements. For topology III,(Fig.3.6c) similar results were observed as per topology I. This is because the links in Topology III were not saturated with network traffic.

Fig.3.6c presents the transmission of TCP traffic in topology III. In contrast to from a fixed download size (Figs.3.6a and 3.6b), the TCP traffic file downloading from iPerf was run with a time constraint of 1200s. Again *SDN+P4* (red) achieves a higher throughput than that achieved with *SDN+OvS* (blue), demonstrating that P4 switches process packet forwarding in *SDN+P4* faster than OvS switches in *SDN+OvS*.

Table 3.3 summarises the performance metrics in terms of throughput, packet loss, Syn packet delay, delay, amount of data transmitted and transmission time for this group of

(a) Network performance of TCP transfer in simple multi-path topology (topology I)

	Topology I - Single Type of Traffic (TCP)		
	SDN+OvS	SDN+P4	Improvement
Throughput	1.09*10^7bps	1.19*10^7bps	9.17%
Packet Loss	4%	0.75%	-81.25%
Delay	0.0041s	0.0035s	-14.6%
Syn Delay	0.43 ms	0.096 ms	-77.6%
Data transmitted	1GB	1GB	N/A
Total transmission time	822.8s	752.1s	-8.5%

(b) Network performance of TCP transfer in simple-grid topology (topology II)

	Topology II - Single Type of Traffic (TCP)		
	SDN+OvS	SDN+P4	Improvement
Throughput	1.05*10^7bps	1.175*10^7bps	11.9%
Packet Loss	5.9%	3.9%	-33.8%
Delay	0.06s	0.03s	-50%
Syn Delay	0.94ms	0.05 ms	-94.68%
Data transmitted	1GB	1GB	N/A
Total transmission time	820.2s	746.1s	-9.034%

(c) Network performance of TCP transfer in the Internet topology (topology III)

	Topology III - Single Type of Traffic (TCP)		
	SDN+OvS	SDN+P4	Improvement
Throughput	1.04*10^7bps	1.169*10^7bps	12.4%
Packet Loss	1.8%	3.7%	-51.35%
Delay	2.87s	0.94s	-67.24%
Syn Delay	3.89ms	1.4 ms	-64.01%
Data transmitted	1.0796GB	1.2106GB	12.13%
Total transmission time	1200s	1200s	N/A

Table 3.3 Network performance for case study 2 - TCP traffic

experiments, averaged over the connections for each respective topology in Table 3.2. In all counts, *SDN+P4* has shown significant improvement over *SDN+OvS*. Throughput has shown improvement in topology I, II and III by 9%, 11% and 12% respectively. Syn delay has also been reduced with SDN+P4, by 77%, 94% and 64% respectively, for the three topologies. The TCP Syn Delay [189] has been calculated by determining which switch architecture can complete the TCP handshake more efficiently and effectively to establish a TCP connection. Given the nature of the TCP traffic, SYN packets play a crucial rule in establishing a TCP connection. As shown in Table 3.3, *SDN+P4* has spent the least amount of time in synchronising the stateful connection between the respective client and server. Having achieved a higher throughput for all cases in *SDN+P4*, total transmission time has been reduced by 8% and 9% respectively, for topologies I and II respectively with a 12% increase on the total data transmitted for topology III. The application of *SDN+P4* has improved the performance of the network significantly.

3.4.3 Case Study 3 - UDP

In case study 3, UDP traffic was generated via file downloads using iPerf. Clients downloaded a file of 1GB from respective servers for topologies I and II with a transmission of UDP for 1200s for topology III. The results of the experiments are presented in Fig.3.7 together with a summary of the results given in Table 3.4.

For all three experiments, Figs.3.7a, 3.7b and 3.7c show that *SDN+P4* achieved a higher and constant throughput in comparison to *SDN+OvS*. The initial delay in *SDN+OvS* was longer than for *SDN+P4*, because the route discovery delay is significant in *SDN+OvS* but lesser with *SDN+P4*. With successful population of FlowRules, consuming a greater time for route discovery bottlenecks can result for real-time applications that involve a demanding service such as, for example Vehicle-to-Everything (V2X).

In Fig.3.7a, the fixed size UDP download completed earlier for *SDN+P4* in approximately 510s (Fig.3.7a), than for SDN+OvS which took approximately 640s, (Fig.3.7b). In both *SDN+P4* and *SDN+OvS*, traffic traversed the same route for topologies I and II. In contrast to Fig.3.7a and 3.7b, the UDP traffic file download in iPerf ran for a fixed 1200s. As shown in Fig.3.7c, *SDN+P4* (red) achieved a higher throughput than *SDN+OvS* (blue). In all cases the P4 switches, process packet forwarding in *SDN+P4* faster than non-P4 switches in *SDN+OvS*.

For case study 3, Figs.3.8a, 3.8b and 3.8c present jitter for UDP packets. As observable by the Figs.3.8a, 3.8b, jitter in *SDN+OvS* (blue lines) have recorded a lower jitter than that of collated in *SDN+P4*. Albeit being recorded for a fraction of a few seconds in a quasi-periodic trend, an argument can be presented that these should not contribute towards an application that requires stringent jitter constraints. The primary reason for this quasi-periodic pattern of higher jitter is the higher throughput that was recorded by *SDN+P4*. This has been corroborated by the authors at [20]. In Table 3.4a, *SDN+OvS* recorded an average jitter of 0.298ms while *SDN+P4* produced an average of 0.231 ms using the equation (1).

$$\frac{1}{M}\sum_{f=1}^{M}\sum_{i=1}^{N_f} J_{fi}/N_f \tag{3.1}$$

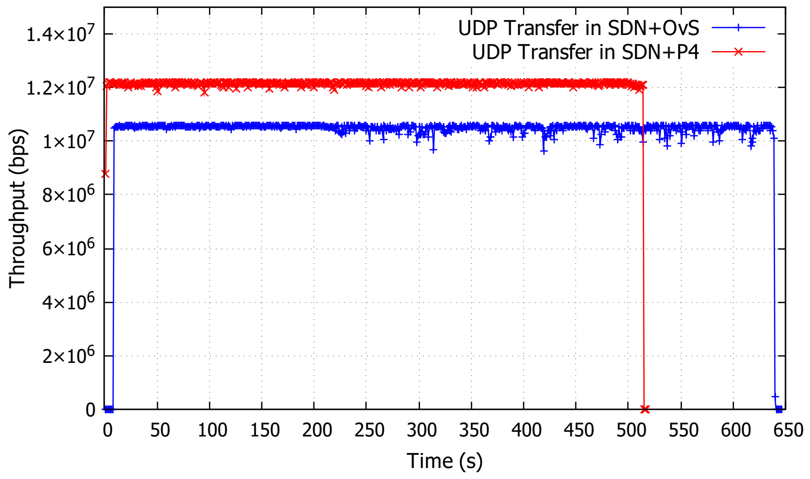
Here M represents the number of flows in the network, N_f denotes the number of packets in flow f and J_{fi} represents the jitter in packet i of flow f. With the application of equation (1) and the analysis of jitter, *SDN+P4* recorded a reduction in jitter of 22% in comparison to *SDN+OvS*.

Table 3.4 summarises the performance metrics in terms of throughput, packet loss, delay, amount of data transmitted and transmission time for this group of experiments, averaged over the respective connections given in Table 3.2. In all cases, *SDN+P4* has demonstrated a significant improvement in network related statistics in comparison to *SDN+OvS*. For example, the grid topology has demonstrated a 13% improvement of throughput in *SDN+P4* in comparison to *SDN+OvS*. *SDN+P4* has reduced the packet loss from 22% to 9% a reduction of 59%. Recorded delay for UDP traffic has reduced from 1.95s for *SDN+OvS* to 0.32s for *SDN+P4*, an increment of 83%. Lastly, the total transmission time has reduced by 14% due to the faster download speed achieved by *SDN+P4* over that achieved with *SDN+OvS*. Although the above results for each case study were conducted on software switches, the experiments can be conducted on hardware switches to further support our hypothesis with improved results [29].

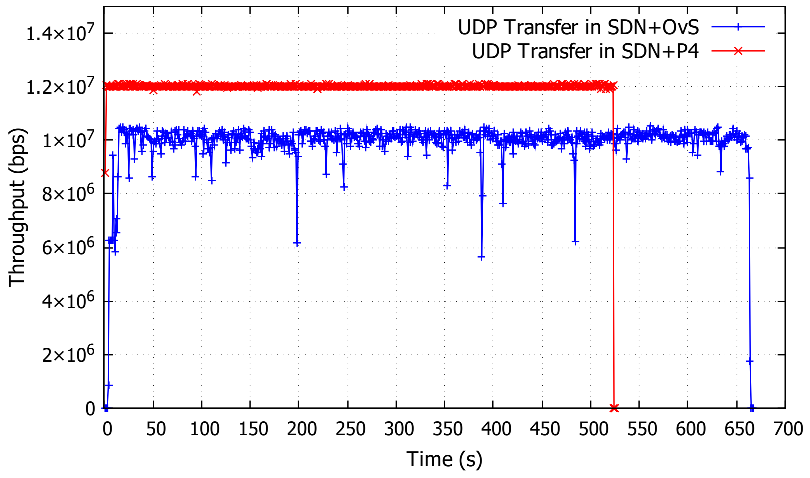
3.4.4 Case Study 4- Content Delivery Network

In this case study we used VLC-player for our experiments. Ability to program the video streaming via a Python script, lightweight deployment and the ability to collect data for video traffic were the rationale for the selection of VLC-player. Fig.3.9 presents the captured data of a video stream for topologies I,II and III.

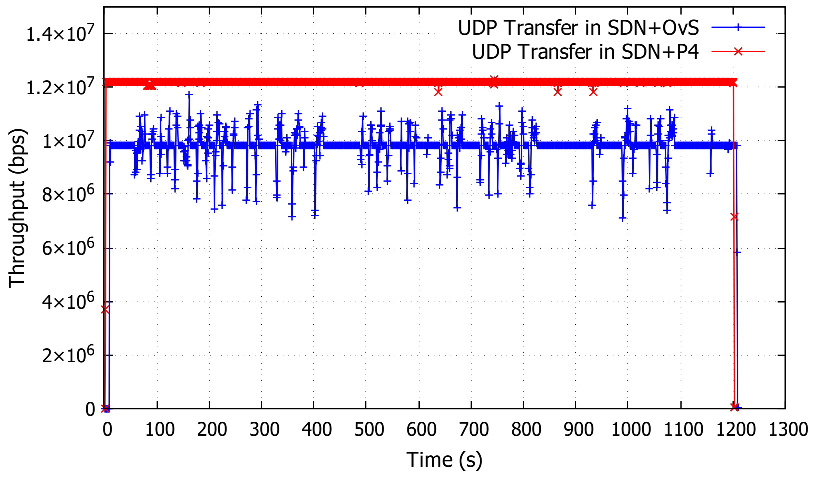
Figs.3.9a, 3.9b and 3.9c show that for each of the three topologies, the video for *SDN+P4* had a higher throughput than with *SDN+OvS*. Table 3.5 shows further that, throughput has improved by 60% in the multi-path topology, 77% in the grid topology and 60% in the Internet topology for *SDN+P4* in comparison to *SDN+OvS*. The high throughput achieved with *SDN+P4* topologies, resulted in a higher amount of data being transferred between hosts. As shown in Table 3.5, the amount of data transmitted improved by 46% in the multi-path topology, 50% in the grid topology and 43% in the Internet topology for *SDN+P4*



(a) UDP transfer of 1GB data using iPerf in topology I



(b) UDP transfer of 1GB data using iPerf in topology II



(c) UDP transfer for 1200s using iPerf in topology III

Fig. 3.7 Throughput of UDP data transfer in case study 3

(a) Network performance of UDP transfer in simple multi-path topology (topology I)

	Topology I - Single Type of Traffic (UDP)		
	SDN+OvS	*SDN+P4*	Improvement
Throughput	1.06*10^7bps	1.2*10^7bps	13%
Packet Loss	22%	9%	(-59%)
Delay	1.9519s	0.3221s	(-83%)
Jitter	0.298 ms	0.231 ms	(-22%)
Data transmitted	1GB	1GB	N/A
Total transmission time	640s	514s	(-14%)

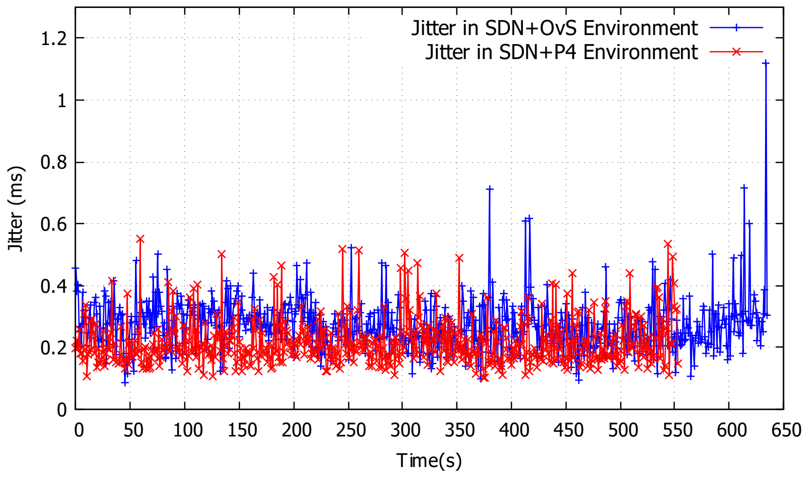
(b) Network performance of UDP transfer in simple-grid topology (topology II)

	Topology II - Single Type of Traffic (UDP)		
	SDN+OvS	*SDN+P4*	Improvement
Throughput	1.04*10^7bps	1.2*10^7bps	15%
Packet Loss	37%	7%	(-81%)
Delay	1.9344s	0.3744s	(-80%)
Jitter	0.371 ms	0.204 ms	(-45%)
Data transmitted	1GB	1GB	N/A
Total transmission time	670s	524s	(-17%)

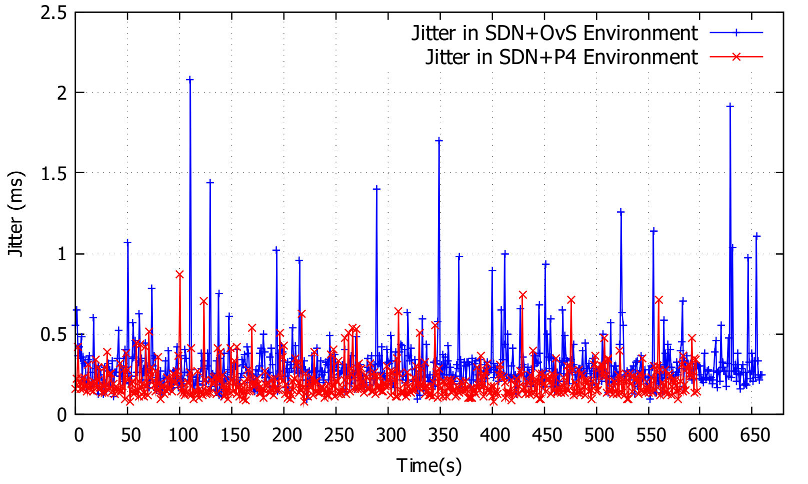
(c) Network performance of UDP transfer in the Internet topology (topology III)

	Topology III - Single Type of Traffic (UDP)		
	SDN+OvS	*SDN+P4*	Improvement
Throughput	0.98*10^7bps	1.2*10^7bps	22%
Packet Loss	39%	8%	(-79%)
Delay	1.9905s	1.0432s	(-60%)
Jitter	0.301 ms	0.2007 ms	(-33.3%)
Data transmitted	1.4723GB	1.6178GB	10%
Total transmission time	1200s	1200s	N/A

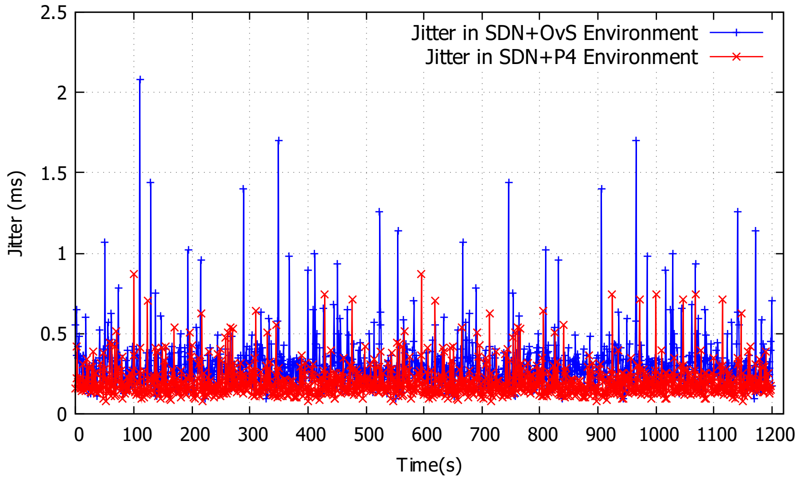
Table 3.4 Network performance for case study 3 - UDP traffic



(a) Jitter experienced for UDP Transmission in Topology I

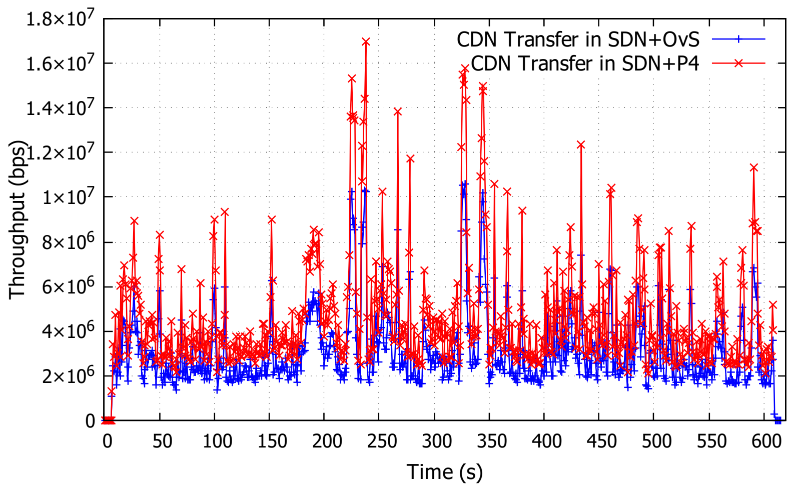


(b) Jitter experienced for UDP Transmission in Topology II

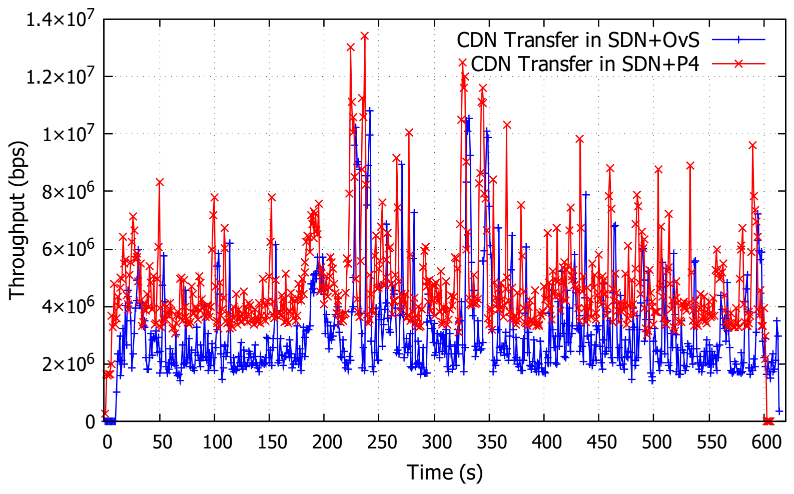


(c) Jitter experienced for UDP Transmission in Topology III

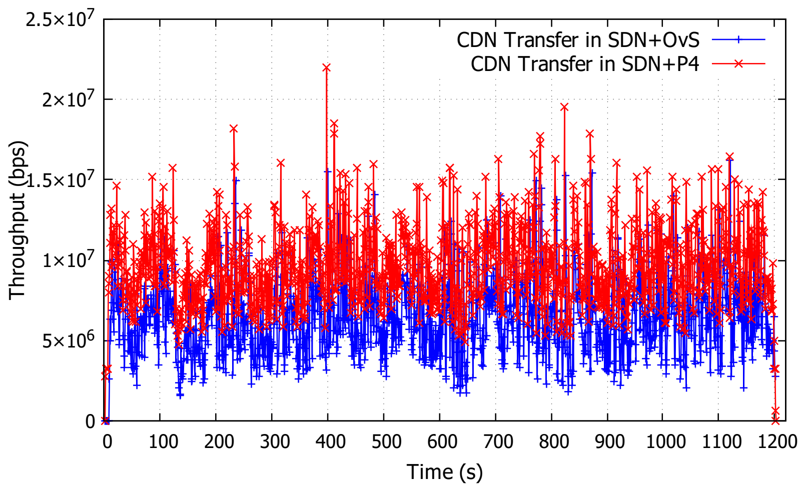
Fig. 3.8 Jitter of UDP data transmission in case study 3



(a) Live video stream in topology I for 600s

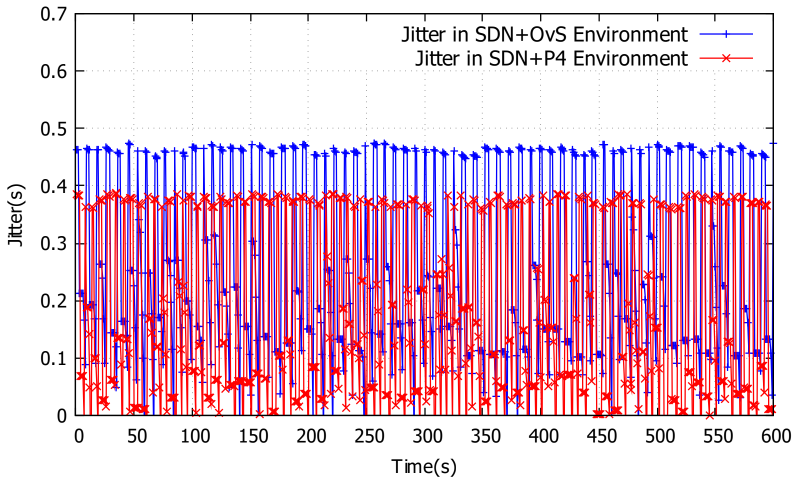


(b) Live video stream in topology II for 600sI

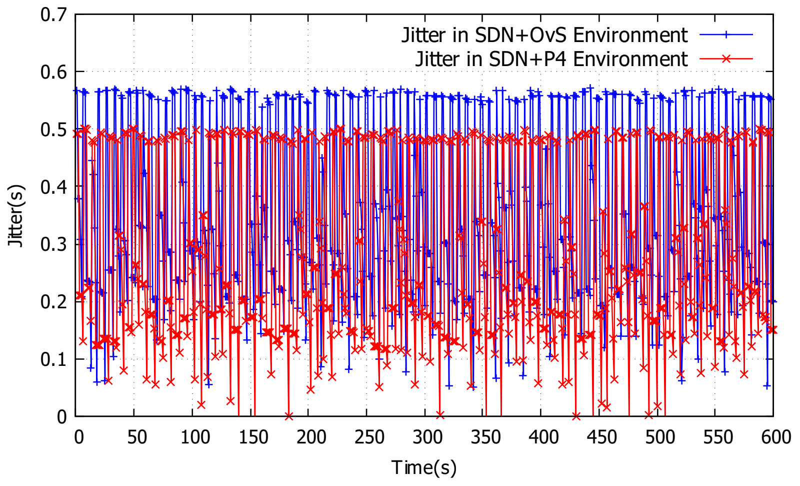


(c) Live video stream in topology III for 1200s

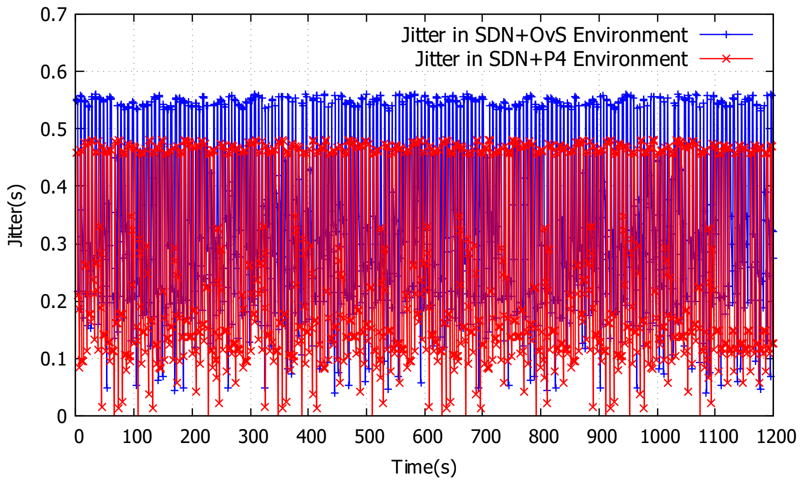
Fig. 3.9 Throughput of video content transfer in case study 4



(a) Jitter experienced in CDN Transmission in Topology I



(b) Jitter experienced in CDN Transmission in Topology II



(c) Jitter experienced in CDN Transmission in Topology III

Fig. 3.10 Jitter of CDN data transmission in case study 4

(a) Network performance of CDN transfer in simple multi-path topology (topology(I))

	Topology I - Single Type of Traffic (CDN)		
	SDN+OvS	SDN+P4	Improvement
Throughput	1.0*10^6bps	1.6*10^6bps	60%
Data transmitted	667.63MB	973.61MB	46%
Buffering delay	8.672s	2.7s	(-69%)
Jitter	0.255ms	0.164ms	(-35.6%)

(b) Network performance of CDN transfer in simple-grid topology (topology(II))

	Topology II - Single Type of Traffic (CDN)		
	SDN+OvS	SDN+P4	Improvement
Throughput	0.9*10^6bps	1.6*10^6bps	77%
Data transmitted	647.63MB	973.61MB	50%
Buffering delay	16.9s	2.7s	(-84%)
Jitter	0.351ms	0.207ms	(-41%)

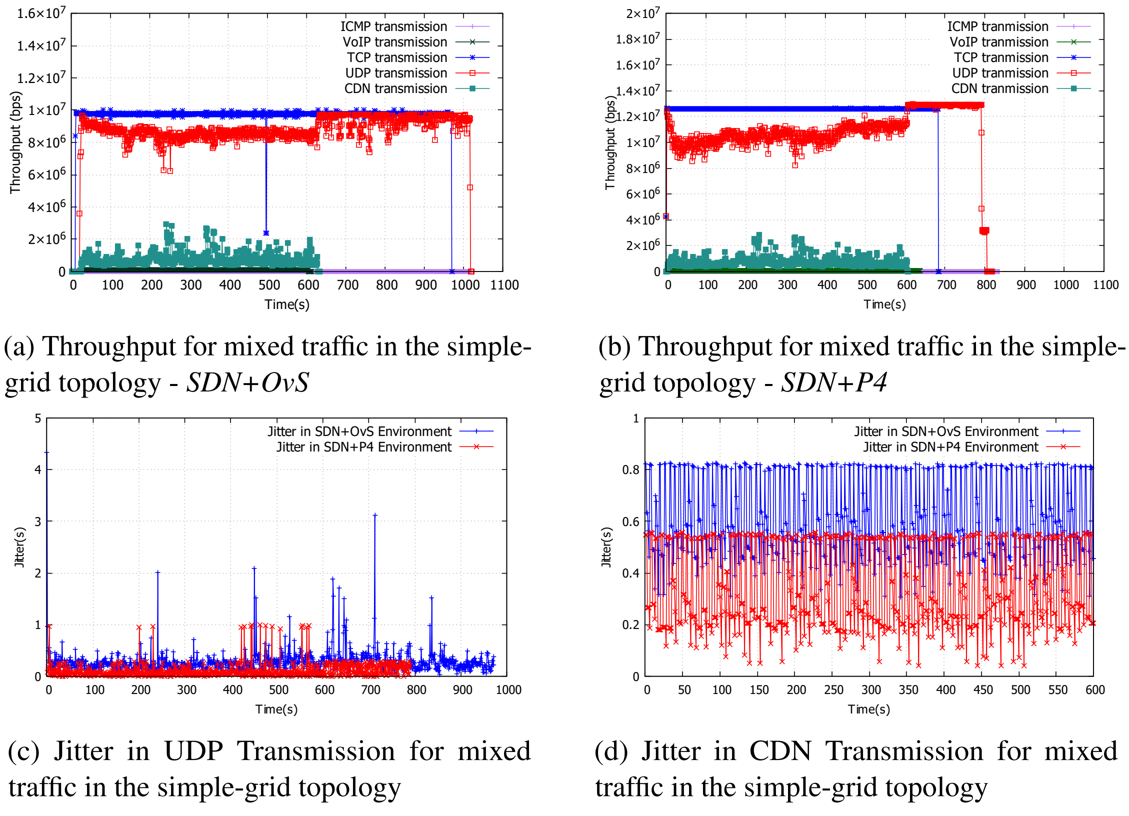
(c) Network performance of CDN transfer in the Internet topology (topology(III))

	Topology III - Single Type of Traffic (CDN)		
	SDN+OvS	SDN+P4	Improvement
Throughput	1.0*10^6bps	1.6*10^6bps	60%
Data transmitted	1609.17MB	2308.88MB	43%
Buffering delay	12.94s	2.17s	(-83%)
Jitter	0.342ms	0.261ms	(-41%)

Table 3.5 Network performance for case study 4 - CDN traffic

in comparison to *SDN+OvS*. A notable delay of initiating the video stream can be observed at Figs.3.9a, 3.9b and 3.9c for *SDN+OvS* (blue line). This again is due to the delay caused in route discovery, resulting in the delayed start of the video terminating later in *SDN+OvS*. This can be seen by observing the averaged buffering delay illustrated in Table 3.5. Buffering delay in *SDN+P4* has been reduced by 69%, 84% and 83% for topologies I, II and III respectively in comparison to *SDN+OvS*. Figs.3.10a, 3.10b and 3.10c represent the observed jitter during the CDN transmission. The equation at (1) was used once more to calculate the averaged values for jitter. As shown in Table 3.5, jitter in *SDN+P4* has been reduced by 35%, 41% and 23% respectively in comparison to *SDN+OvS* for topologies I, II and III. In our experiments, topologies I and II streamed a video for 600s whilst topology III streamed a video for 1200s. Although the streaming times are different, the performance gain remained constant across topologies I, II and III.

3.5 Results and Analysis of Tier-II - Multiple Types of Traffic Running Simultaneously



(a) Throughput for mixed traffic in the simple-grid topology - *SDN+OvS*

(b) Throughput for mixed traffic in the simple-grid topology - *SDN+P4*

(c) Jitter in UDP Transmission for mixed traffic in the simple-grid topology

(d) Jitter in CDN Transmission for mixed traffic in the simple-grid topology

Fig. 3.11 Data for mixed traffic over the grid topology (topology II) - case study 5

Having investigated the performance of single types of traffic running over different network topologies for *SDN+OvS* and *SDN+P4*, we now study the performance of mixed types of traffic running simultaneously over topologies II and III.

3.5.1 Case Study 5 - Mixed Type of Traffic over Topology II

Five types of traffic from Table 3.2, ICMP, VoIP, TCP, UDP and CDN, were run simultaneously. The performance of each type of traffic is presented in Table 3.6. Table 3.6a (ICMP traffic) reports an improvement of 6.7% in throughput for *SDN+P4* in comparison

(a) Performance of ICMP traffic in case study 5

	Topology II - Mixed Traffic (ICMP)		
	SDN+OvS	*SDN + P4*	Improvements
Throughput	5.85*10^3 bps	6.3*10^3 bps	6.7%

(b) Performance of TCP traffic in case study 5

	Topology II - Mixed Traffic (TCP)		
	SDN + OvS	SDN + P4	Improvement
Throughput	1.05*10^7bps	1.18*10^7bps	12%
Packet Loss	7.9%	5%	(-36%)
Delay	3.1 s	2.9s	(-6%)
Syn Delay	2.6 ms	0.6 ms	(-77%)
Data transmitted	1 GB	1 GB	N/A
Total transmission time	982.1 s	807 s	(-18%)

(c) Performance of UDP traffic in case study 5

	Topology II - Mixed Traffic (UDP)		
	SDN+OvS	*SDN+P4*	Improvement
Throughput	7*10^6bps	1*10^7bps	43%
Packet Loss	38%	10%	(-73%)
Delay	1.9157s	0.4064s	(-78%)
Jitter	0.480 ms	0.247 ms	(-48%)
Total transmission time	1020s	800s	(-21%)

(d) Performance of CDN traffic in case study 5

	Topology II - Mixed Traffic (CDN)		
	SDN+OvS	SDN+P4	Improvement
Throughput	3.8*10^5bps	4*10^5bps	5%
Data transmitted	588.41 MB	817.54 MB	38%
Buffering delay	3.11s	1.91s	(-38%)
Jitter	0.351ms	0.33ms	(-5%)

Table 3.6 Network Performance of for case study 5

to the throughput achieved by *SDN+OvS*. Table 3.6b (TCP traffic) records an improvement of 12% in throughput from 1.05*10^7bps for *SDN+OvS* to 1.18*10^7 bps for *SDN+P4*. These are represented in Fig.3.11a and Fig.3.11b by the blue line. Packet loss, packet delay, synchronisation delay and total transmission time achieved significant reductions for *SDN+P4*over *SDN+OvS* of, 36%, 6%, 77% and 18% respectively.

Table 3.6c (UDP traffic) reports an increase in UDP throughput of 43% (from 7*10^6bps in *SDN+OvS* to 1.0*10^7bps in *SDN+P4*) as illustrated in Figs.3.11a and 3.11b by the red line. It was also observed that *SDN+P4* completed the download of 1GB of UDP traffic in 800s in contrast to *SDN+OvS* which took 1020 seconds. Packet loss, delay and jitter were each observed to have reduced values for SDN+P4 in comparison to those obtained using SDN + OvS, these being 73%, 78% and 48% respectively. The reduction in jitter for UDP packets is shown in Fig.3.11c.

Table 3.6d (CDN traffic) records an increase of 5% in throughput from 3.8*10^5bps for *SDN+OvS* to 4*10^5bps for *SDN+P4*. This is illustrated by the green line in Figs.3.11a and 3.11b. Higher throughput resulted in *SDN+P4* transferring more data (817.54MB) in comparison to *SDN+OvS* (588.41MB), an improvement of 38% for the same video file. Buffer delay and jitter in *SDN+P4* resulted in significant reductions of 38% and 5% respectively, in contrast to *SDN+OvS*. The reduction in jitter is shown in Fig.3.11d.

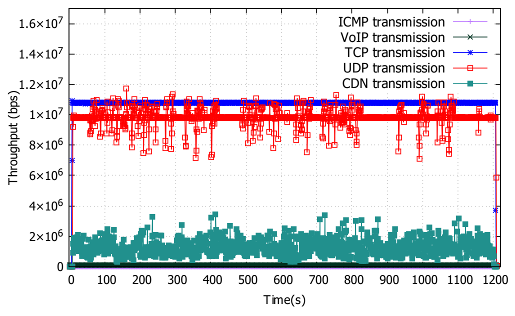
The difference in throughput has an effect on the quality of the video. Fig.3.13a shows a comparison for the 34th frame captured during the experiments for both *SDN+OvS* and *SDN+P4*. The same frame was sent from the server side, with the client side in *SDN+OvS* displaying a distorted frame in comparison to that received by the client with *SDN+P4*.

3.5.2 Case Study 6 - Simultaneous Run over Topology III

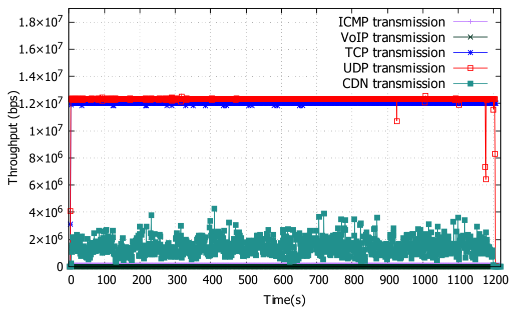
Table 3.7 presents the data collected during this case study. Table 3.7a, the throughput of ICMP has seen an improvement of 6.7% in *SDN+P4* in comparison to *SDN+OvS*.

Table 3.7b (TCP traffic) records an increase in TCP throughput of 19% from 0.99*10^7bps in *SDN+OvS* to 1.18*10^7bps in *SDN+P4*. They are illustrated by the blue lines in Figs.3.12a and 3.12b. Packet loss, packet delay and synchronisation delay in *SDN+P4* reflects significant reductions in comparison to that achieved using *SDN+OvS* of 75%, 65% and 45% respectively. The data transmitted achieved a 23% improvement from 0.82GB for *SDN+OvS* to 1.01GB with *SDN+P4*.

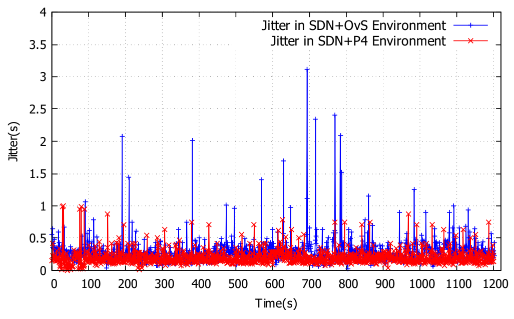
Table 3.7c (UDP traffic) records an increase in UDP throughput of 33% from 9*10^6bps in *SDN+OvS* to 1.2*10^7bps in *SDN+P4*. They are illustrated by the red lines in Figs.3.12a and 3.12b. Packet loss, delay and jitter in *SDN+P4* reflects significant reductions in comparison to that achieved using *SDN+OvS* of 69%, 58% and 31% respectively. A reduction in jitter for UDP packets is shown in Fig.3.12c. The higher speed, reduced delay, jitter and packet loss in *SDN+P4* result in an increase in data transmission (1.602GB) for *SDN+P4* in comparison to (1.456GB) for *SDN+OvS*.



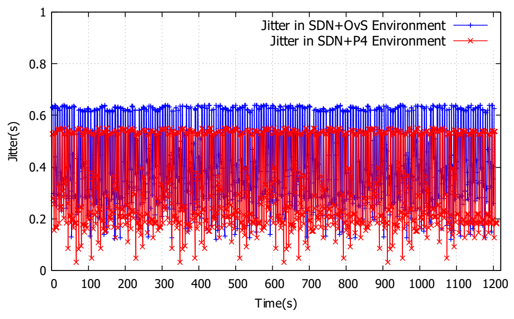
(a) Mixed traffic in the Internet topology *SDN+OvS*



(b) Mixed traffic in the Internet topology *SDN+P4*



(c) Jitter experienced in UDP Transmission while simultaneous applications occupy the network space in Topology III



(d) Jitter experienced in CDN Transmission while simultaneous applications occupy the network space in Topology III

Fig. 3.12 Data of mixed traffic over the Internet topology (topology III) - case study 6

Table 3.7d (CDN traffic) shows an increase in throughput of 44% from $0.97*10^6$bps in *SDN+OvS* to $1.4*10^6$bps in *SDN+P4*. This is illustrated by the green line in Figs.3.12a and 3.12b. Higher throughput resulted in *SDN+P4* transferring more data (1451.3MB) in comparison to *SDN+OvS* (1183.4MB), an improvement of 23% for the same video file. Buffer delay and jitter in *SDN+P4* resulted in significant reductions of 61% and 21% respectively, in comparison to *SDN+OvS*. The reduction in jitter is shown in Fig.3.12d.

The difference in throughput has directly affected the quality of the video. Fig.3.13b shows a comparison for the 34^{th} frame captured during the experiments for both *SDN+OvS* and *SDN+P4*. The same frame was sent from the server side. At the client side *SDN+OvS* displayed the previous frame (33^{rd}) in comparison to that received by the client via *SDN+P4* (34^{th}).

(a) Performance of ICMP traffic in case study 6

	Topology III - Mixed Traffic (ICMP)		
	SDN+OvS	*SDN + P4*	Improvements
Throughput	5.9*10^3bps	6.3*10^3bps	6.7%

(b) Performance of TCP traffic in case study 6

	Topology III - Mixed Traffic (TCP)		
	SDN + OvS	SDN + P4	Improvement
Throughput	0.99*10^7bps	1.18*10^7bps	19%
Packet Loss	0.4%	0.1 %	(-75%)
Delay	3.2s	1.1s	(-65%)
Syn Delay	3.84ms	2.1ms	(-45%)
Data transmitted	0.82GB	1.01GB	23%

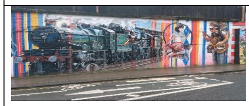
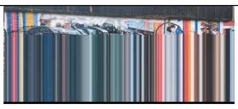
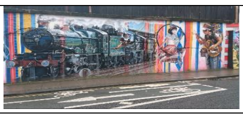
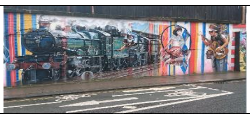
(c) Performance of UDP traffic in case study 6

	Topology III - Mixed Traffic (UDP)		
	SDN+OvS	*SDN+P4*	Improvement
Throughput	9*10^6bps	1.2*10^7bps	33%
Packet Loss	36%	11%	(-69%)
Delay	2.516s	1.034s	(-58%)
Jitter	0.292ms	0.201ms	(-31%)
Data transmitted	1.456GB	1.602GB	10%

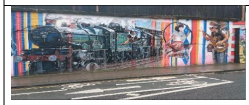
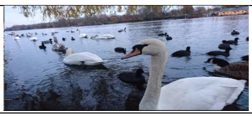
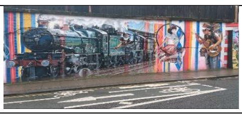
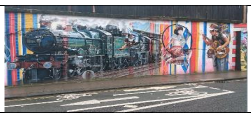
(d) Performance of CDN traffic in case study 6

	Topology III - Mixed Traffic (CDN)		
	SDN+OvS	SDN+P4	Improvement
Throughput	0.97*10^6bps	1.4*10^6bps	44%
Data transmitted	1183.4MB	1451.3MB	23%
Buffering delay	20s	7.8s	(-61%)
Jitter	0.422ms	0.331ms	(-21%)

Table 3.7 Network performance for case study 6

Topology II – Mixed Traffic (CDN)				
SDN+OvS			***SDN+P4***	
Server		**Client**	**Server**	**Client**
				
Observing time	**90.0s**	**90.0s**	**90.0s**	**90.0s**
Sequence No of the Frame observed	**34th**	**34th**	**34th**	**34th**

(a)

Topology III – Mixed Traffic (CDN)				
SDN+OvS			***SDN+P4***	
Server		**Client**	**Server**	**Client**
				
Observing time	**90.0s**	**90.0s**	**90.0s**	**90.0s**
Sequence No of the Frame observed	**34th**	**33rd**	**34th**	**34th**

(b)

Fig. 3.13 Comparison of CDN frames in internet topology for both *SDN+OvS* and *SDN+P4* for case studies 5 and 6.

3.6 Discussion

In the context of this research, we explored the performance of SDN with *OvS* (*SDN+OvS*) and SDN with *P4* (*SDN+P4*). We have seen that *SDN+P4* outperforms *SDN+OvS* for each of our case studies using different types of traffic and topologies. In this section, we present a discussion based on our research questions.

We investigated the overhead created due to the slow path utilisation of OvS and the performance variation in comparison to a P4 target switch. With the evolution of the internet and the increased number of connected devices, networks will face congestion. With more and more packets requiring processing using a controller, a network model that utilises a slow path approach such as OvS will potentially lead to an exponential growth in traffic congestion. Our results suggest that the network can maintain a gain in performance whilst links are saturated with traffic for the *SDN+P4* environment in comparison to that experienced with *SDN+OvS*.

We evaluated the performance of networks when *SDN+P4* is employed rather than *SDN+OvS*. The evolution of 5G and beyond has led to the need to evaluate methods for reducing the delay at the core. Initialising programmability in the network has been shown to increase performance at the core. To the best of our knowledge, current literature does not evaluate the performance of the network when the control plane and data plane programmability (*SDN+P4*) is employed in comparison to control plane (*SDN+OvS*) programmability. Fig 3.3 (III) illustrates the Internet topology which we employed for the purpose of this research. Results were collected over the emulation of a core network (Transit-stub models).

We evaluated the performance of applications where *SDN+P4* has been employed in contrast to *SDN+OvS*. For a time-sensitive application with minimal latency requirements, reducing the delay at the core can be essential. For example Vehicle-to-Vehicle (V2V) and Ultra Reliable Low Latency Communication (URLLC) applications. A solution, that processes packets in parallel as opposed to sequential processing in OvS, has been considered in this research and its effect on performance in applications. Our research has established that with the initialisation of *SDN+P4* with parallel processing of packets, various applications have better performance in comparison to applications run over *SDN+OvS*. We also evaluated the quality of applications that occurred due to faster processing achieved with *SDN+P4* in comparison to *SDN+OvS*. The statistics such as increased bps and throughput, reduced delay jitter, packet loss, delay and buffering time have led to a higher quality of application at the receiver's end. Improvement of the quality of the video at the receiver end for a video streaming application (Fig.3.13) has also been presented to confirm our contributions.

Furthermore, we investigated whether the type of traffic has a bearing on performance for *SDN+P4* in comparison to *SDN+OvS*. Given that the majority of network traffic will be accumulated by multimedia applications in the future, a fair switching mechanism that enhances the performance of such applications will provide a beneficial factor for service providers. As the majority of the downloaded traffic in a modern context consists of UDP traffic (multimedia applications) and the demand for such applications is expected to grow exponentially in the future, the employment of *SDN+P4* is a viable solution for service providers looking to provide an improved service. Protocols such as Google Quick UDP Internet Connections (GQUIC) in mobile telecommunication will also benefit from the employment of *SDN+P4*. The employment of *SDN+P4* will also benefit non-media applications where a connection-less protocol is employed by a service provider. For instance, communication between a Radio Device and eNB, Multi-access Edge Clouds (MEC) or between Radio Access Networks (RAN) and Fog nodes [178] can benefit through the use of *SDN+P4*. To meet the requirements for faster convergence and faster processing of packets, *SDN+P4* will serve as an inventive solution. A detailed discussion on the testbed and the rationality for the employment can be found in Chapter 4.

3.7 Chapter Summary

The above Chapter has answered the research questions given at the beginning of this study. We conclude that the application of *SDN+P4* has enabled networks to improve their performance for different topologies and traffic. The results from our experiments show an improvement in, for example, delay and packet loss which has reduced significantly for *SDN+P4*, whilst throughput has increased for all case studies. *SDN+P4* has improved performance in the networks through the use of parallel processing in P4, which has complemented the standardised SDN architecture. The results for all case studies indicate that *SDN+P4* is a promising alternative to *SDN+OvS* providing a resilient approach for future networking.

Through a combination of SDN and P4 (control plane programmability with data plane programmability), we have established that it is possible to provide an improved service to clients with *SDN+P4* than with *SDN+OvS*. Across all case studies, parallel processing in P4 has provided an increase in available queues for processing traffic with the utilisation of a flexible parser. With the evolution of the internet and with the heterogeneity of applications, *SDN+P4* will, we believe, provide an improved service to all use cases for 5G and beyond. Based on the results collated from the conducted experiments of this chapter, we implemented 'data plane programmability' (P4) into our 5G mobile telecommunication testbed connecting our RAN network to the Core Network, with the aim of improving performance. The programmability at the data plane is implemented using a BMv2 switch on Intel 10GbE x520 NIC, employed at the edge of the deployed 5G-MEC mobile telecommunication testbed, due to the benefits (traffic engineering, in-band telemetry for latency critical services, and Cyber security) as emphasized by [190]. The details and the configurations of the 5G mobile telecommunication testbed, will follow in the next chapter, where we aim to answer the research question: How do we design and build a testbed for 5G mobile telecommunications?

Chapter 4

A Real-Time 5G Mobile Telecommunication Testbed

This chapter presents the Real-Time testbed that was developed in order to implement a 5G mobile telecommunication testbed. Service providers have now entered the implementation phase for 5G mobile telecommunication networks. With this, the concept of Multi-access Edge Computing (MEC) will play a crucial role when providing services on the go with low latency, high availability and high bandwidth. However, due to the low processing power of MEC nodes, adversaries may target the platform for malevolent purposes. In this chapter, we focus on building a realistic 5G-MEC testbed to run legitimate traffic and network attacks and to collect 5G datasets for 5G-MEC.

4.1 Introduction

5G mobile telecommunication networks are poised to become the key enabler for Information Communication Technology (ICT), accommodating a variety of use cases, case studies, and scenarios that were not covered under predecessor technologies. Use cases such as enhanced mobile broadband, massive machine-type communication, and ultra-reliable low-latency communication will function in a diverse yet distinct manner. The integration of Multi-access Edge Computing (MEC) necessitates a substantial modification at the service-based architectural level to potentially address the diverse requirements of these use cases. The importance of securing this architecture has gained the attention of researchers working in this field, however, their respective research in traffic analysis and intrusion detection for 5G-MEC does not employ publicly available datasets collected from a mobile telecommunication 5G testbed.

Experiments
RQ 2:Implementation of a 5G Mobile Telecommunication Testbeds
1. Study of the existing simulators and emulators
2. Implement a 5G Mobile Telecommunication Testbed
3. Implement programmability in the 5G Mobile Telecommunication Testbed
4. Emulate a MEC network
5. Emulate Network traffic between User and MEC edge node
6. Conduct Malicious Network Attacks
7. Analysis of Key Performance Indicators
8. Creation of Datasets from 5G-MEC network traffic

Table 4.1 Experiments conducted in this Chapter

Also, existing studies, while presenting frameworks for specific use cases, do not incorporate testbeds for their respective work. The incorporation of testbeds, we believe, would enhance the depth of understanding regarding the feasibility of these frameworks and applications. In this chapter, we present a mobile telecommunication testbed capable of transmitting, capturing, and processing various types of 5G mobile traffic, to further the study of security based 5G research.

4.1.1 Research Questions

The research under this chapter was conducted in order to address the following sub-research-questions.

- Which state-of-the-art platforms can be used to develop an appropriate testbed to generate 5G mobile network traffic?
- What configurations of P4 switches should be implemented on the 5G testbed?
- To what extent is the dataset generated by the developed 5G testbed effective for the analysis of 5G-MEC security?

Experiments were designed as shown in Table 4.1, in order to present the contributions of this Chapter.

4.1.2 Contributions

The key contributions from this study:

- A Testbed was developed employing functions of a 5G mobile telecommunication network with fully functioning P4-BMv2 switches in the Radio Access Network and Core Network.
- A dataset was generated from the testbed containing both legitimate and malicious types of traffic found in 5G networks. This dataset can help in mitigating the False Positive rate for an Intrusion Detection System, applied to a 5G mobile telecommunication network.
- The dataset was tested and compared to existing datasets not generated from a 5G testbed.

4.2 A Mobile Telecommunication Testbed

In order to develop a mobile telecommunication testbed we considered the following state-of-the-art deployment. Open Mobile Evolved Core (OMEC), Network Simulator 3 (NS3) 5G-LENA project, Omnet++ and Mininet-wifi. The high resource requirements for OMEC, the simulation nature of NS3 and Omnet++ and the missing LTE functions of Mininet-wifi made these options inappropriate. A better option we believe, was the OpenAirInterface (OAI) which we employed throughout this research.

OpenAirInterface [86] and [87] is an open-source development managed by the OpenAirInterface Software Alliance. It enables researchers to test and evaluate 5G (and beyond) case studies and scenarios whilst maintaining compliance to the 3GPP standards for both core networks (CN) and radio access networks (RAN).

Being an open-source project, OpenAirInterface provide the user with options, software, and configurations to customise the environment to launch a testbed which caters to their case study. As OpenAirInterface is an open-source project, it can significantly reduce the costs associated with developing and testing wireless communication solutions. OpenAirInterface also has an active and engaging community that supports learning and research. This is advantageous as it will enable researchers to seek community support and guidance. Finally, OpenAirInterface supports rapid prototyping allowing research to progress with new applications and technologies.

In this research, OAI was used in emulating CN's consisting of the mobility management entity (MME), home subscriber server (HSS), the control and user plane separation serving gateway (SPGW-C/U and PGW-C). For this research, OAI was installed on a Ubuntu Bionic distribution with 64GB of RAM running on a Core i7 CPU with 3.4GHz. Hyper-threading[1],

[1]A technology provided by Intel®which allowed more than one thread to run on each core

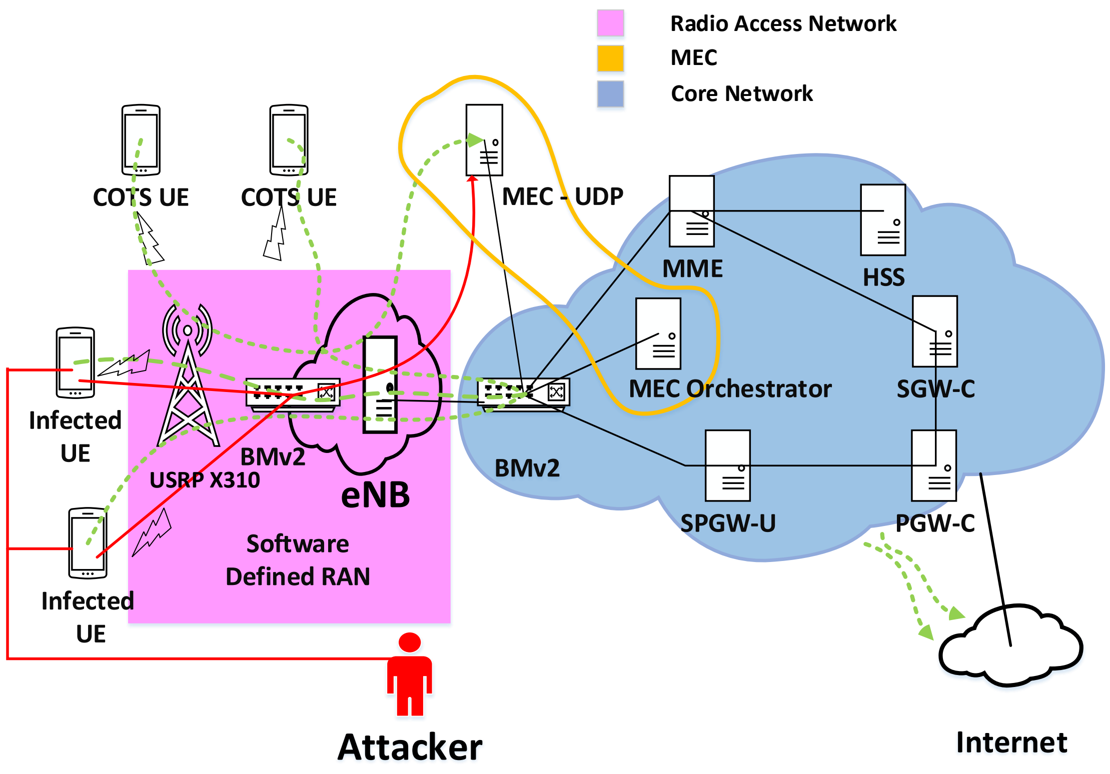


Fig. 4.1 Testbed used to generate datasets for 5G-MEC security analysis

CPU C-States and Speed-Step[2] have been disabled prior initialising the testbed. Our model of a real mobile telecommunication network is shown in Fig.4.1. Screenshots of the system while in operation have been depicted in Figs.4.2 and 4.3

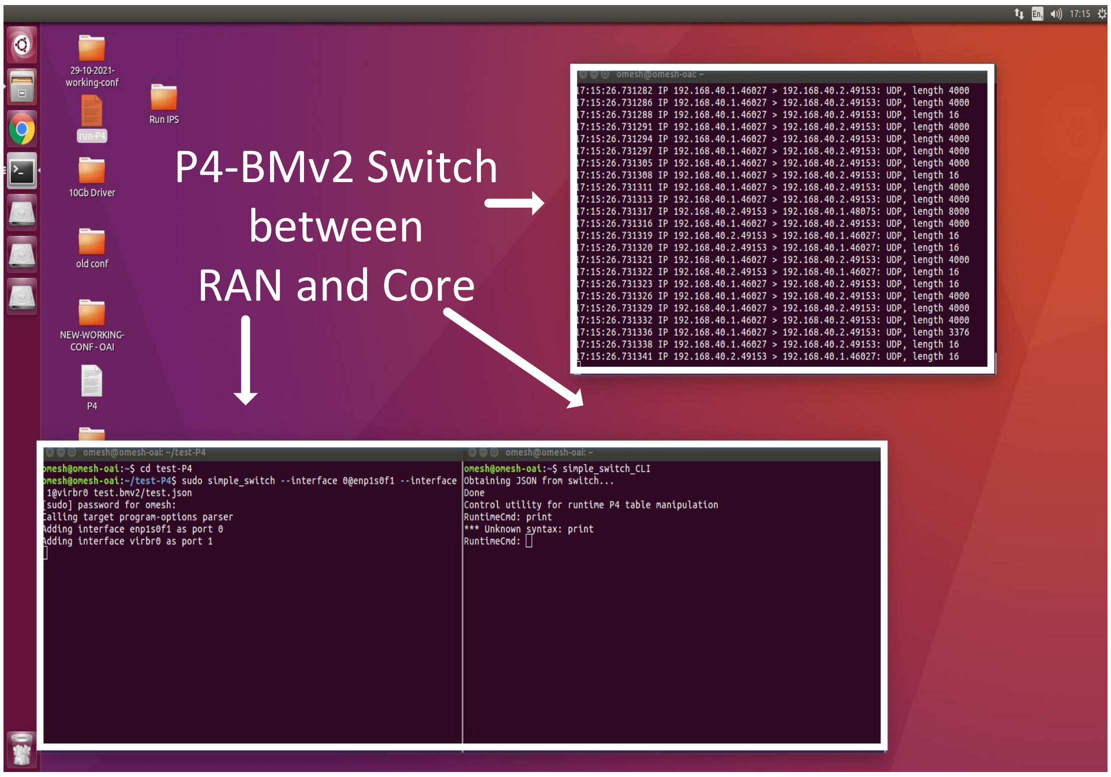
4.2.1 User Equipment (UE)

OpenCells Subscriber Identity Modules (SIM) cards [191] were programmed to communicate with the mobile testbed. To connect legitimate and malicious users, Huawei E3372 Dongles with programmed SIM cards were used to connect to the mobile test-bed.

4.2.2 Radio Access Network (RAN)

RAN is denoted by the purple area in Fig.4.1. To model the Radio Access Network, the OAI core was connected to OAI5g [192] using a Software Defined Radio namely USRP

[2]A technology that allows the clock speed of the processor to be dynamically changed or adjusted by a software

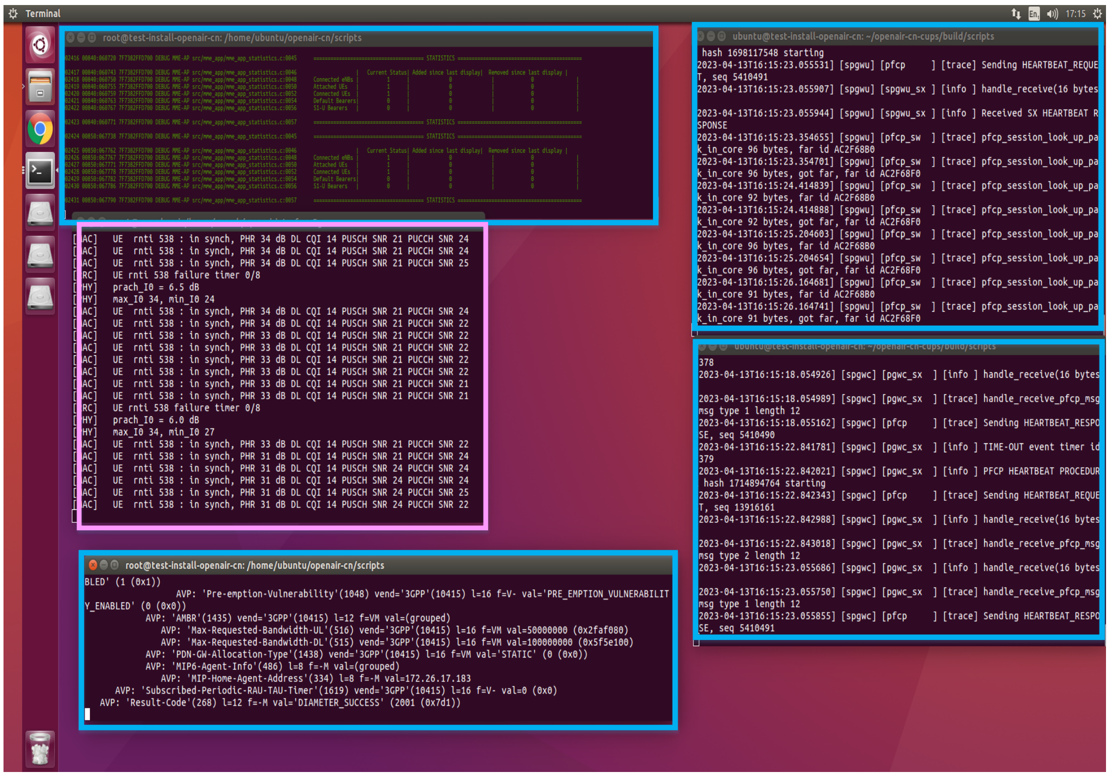


```
omesh@omesh-oai: ~
17:15:26.731282 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731286 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731288 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 16
17:15:26.731291 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731294 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731297 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731305 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731308 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 16
17:15:26.731311 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731313 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731317 IP 192.168.40.2.49153 > 192.168.40.1.48075: UDP, length 8000
17:15:26.731316 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731319 IP 192.168.40.2.49153 > 192.168.40.1.46027: UDP, length 16
17:15:26.731320 IP 192.168.40.2.49153 > 192.168.40.1.46027: UDP, length 16
17:15:26.731321 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731322 IP 192.168.40.2.49153 > 192.168.40.1.46027: UDP, length 16
17:15:26.731323 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 16
17:15:26.731326 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731329 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731332 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 4000
17:15:26.731336 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 3376
17:15:26.731338 IP 192.168.40.1.46027 > 192.168.40.2.49153: UDP, length 16
17:15:26.731341 IP 192.168.40.2.49153 > 192.168.40.1.46027: UDP, length 16
```

```
omesh@omesh-oai: ~/test-P4
omesh@omesh-oai:~$ cd test-P4
omesh@omesh-oai:~/test-P4$ sudo simple_switch --interface 0@enp1s0f1 --interface
 1@virbr0 test.bmv2/test.json
[sudo] password for omesh:
Calling target program-options parser
Adding interface enp1s0f1 as port 0
Adding interface virbr0 as port 1
```

```
omesh@omesh-oai: ~
omesh@omesh-oai:~$ simple_switch_CLI
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: print
*** Unknown syntax: print
RuntimeCmd:
```

Fig. 4.2 P4-BMv2 switch between the RAN and the Core



root@test-install-openair-cn: /home/ubuntu/openair-cn/scripts

```
02416 00840:060720 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0045     ===================================== STATISTICS ============================================

02417 00840:060743 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0046                    |  Current Status| Added since last display|  Removed since last display |
02418 00840:060750 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0048     Connected eNBs |        1       |            0            |            0                |
02419 00840:060755 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0050     Attached UEs   |        1       |            0            |            0                |
02420 00840:060759 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0052     Connected UEs  |        1       |            0            |            0                |
02421 00840:060763 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0054     Default Bearers|        0       |            0            |            0                |
02422 00840:060767 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0056     S1-U Bearers   |        0       |            0            |            0                |

02423 00840:060771 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0057     ===================================== STATISTICS ============================================

02424 00850:067738 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0045     ===================================== STATISTICS ============================================

02425 00850:067762 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0046                    |  Current Status| Added since last display|  Removed since last display |
02426 00850:067767 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0048     Connected eNBs |        1       |            0            |            0                |
02427 00850:067771 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0050     Attached UEs   |        1       |            0            |            0                |
02428 00850:067778 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0052     Connected UEs  |        1       |            0            |            0                |
02429 00850:067782 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0054     Default Bearers|        0       |            0            |            0                |
02430 00850:067786 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0056     S1-U Bearers   |        0       |            0            |            0                |

02431 00850:067790 7F7382FFD700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0057     ===================================== STATISTICS ============================================
```

```
[ AC]   UE  rnti 538 : in synch, PHR 34 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 24
[ AC]   UE  rnti 538 : in synch, PHR 34 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 24
[ AC]   UE  rnti 538 : in synch, PHR 34 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 25
[ RC]   UE rnti 538 failure timer 0/8
[ HY]   prach_I0 = 6.5 dB
[ HY]   max_I0 34, min_I0 24
[ AC]   UE  rnti 538 : in synch, PHR 34 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 24
[ AC]   UE  rnti 538 : in synch, PHR 34 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 22
[ AC]   UE  rnti 538 : in synch, PHR 33 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 22
[ AC]   UE  rnti 538 : in synch, PHR 33 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 22
[ AC]   UE  rnti 538 : in synch, PHR 33 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 22
[ AC]   UE  rnti 538 : in synch, PHR 33 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 22
[ AC]   UE  rnti 538 : in synch, PHR 33 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 21
[ AC]   UE  rnti 538 : in synch, PHR 33 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 21
[ RC]   UE rnti 538 failure timer 0/8
[ HY]   prach_I0 = 6.0 dB
[ HY]   max_I0 34, min_I0 27
[ AC]   UE  rnti 538 : in synch, PHR 33 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 22
[ AC]   UE  rnti 538 : in synch, PHR 31 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 24
[ AC]   UE  rnti 538 : in synch, PHR 31 dB DL CQI 14 PUSCH SNR 24 PUCCH SNR 24
[ AC]   UE  rnti 538 : in synch, PHR 31 dB DL CQI 14 PUSCH SNR 24 PUCCH SNR 24
[ AC]   UE  rnti 538 : in synch, PHR 31 dB DL CQI 14 PUSCH SNR 24 PUCCH SNR 25
[ AC]   UE  rnti 538 : in synch, PHR 31 dB DL CQI 14 PUSCH SNR 24 PUCCH SNR 22
```

root@test-install-openair-cn: /home/ubuntu/openair-cn/scripts

```
BLED' (1 (0x1))
                  AVP: 'Pre-emption-Vulnerability'(1048) vend='3GPP'(10415) l=16 f=V- val='PRE_EMPTION_VULNERABILIT
Y_ENABLED' (0 (0x0))
            AVP: 'AMBR'(1435) vend='3GPP'(10415) l=12 f=VM val=(grouped)
               AVP: 'Max-Requested-Bandwidth-UL'(516) vend='3GPP'(10415) l=16 f=VM val=50000000 (0x2faf080)
               AVP: 'Max-Requested-Bandwidth-DL'(515) vend='3GPP'(10415) l=16 f=VM val=100000000 (0x5f5e100)
            AVP: 'PDN-GW-Allocation-Type'(1438) vend='3GPP'(10415) l=16 f=VM val='STATIC' (0 (0x0))
            AVP: 'MIP6-Agent-Info'(486) l=8 f=-M val=(grouped)
               AVP: 'MIP-Home-Agent-Address'(334) l=8 f=-M val=172.26.17.183
      AVP: 'Subscribed-Periodic-RAU-TAU-Timer'(1619) vend='3GPP'(10415) l=16 f=V- val=0 (0x0)
   AVP: 'Result-Code'(268) l=12 f=-M val='DIAMETER_SUCCESS' (2001 (0x7d1))
```

ubuntu@test-install-openair-cn: ~/openair-cn-cups/build/scripts

```
 hash 1698117548 starting
2023-04-13T16:15:23.055531] [spgwu] [pfcp     ] [trace] Sending HEARTBEAT_REQUE
T, seq 5410491
2023-04-13T16:15:23.055907] [spgwu] [spgwu_sx ] [info ] handle_receive(16 bytes

2023-04-13T16:15:23.055944] [spgwu] [spgwu_sx ] [info ] Received SX HEARTBEAT R
SPONSE
2023-04-13T16:15:23.354655] [spgwu] [pfcp_sw  ] [trace] pfcp_session_look_up_pa
k_in_core 96 bytes, far id AC2F68B0
2023-04-13T16:15:23.354701] [spgwu] [pfcp_sw  ] [trace] pfcp_session_look_up_pa
k_in_core 96 bytes, got far, far id AC2F68F0
2023-04-13T16:15:24.414839] [spgwu] [pfcp_sw  ] [trace] pfcp_session_look_up_pa
k_in_core 92 bytes, far id AC2F68B0
2023-04-13T16:15:24.414888] [spgwu] [pfcp_sw  ] [trace] pfcp_session_look_up_pa
k_in_core 92 bytes, got far, far id AC2F68F0
2023-04-13T16:15:25.204603] [spgwu] [pfcp_sw  ] [trace] pfcp_session_look_up_pa
k_in_core 96 bytes, far id AC2F68B0
2023-04-13T16:15:25.204654] [spgwu] [pfcp_sw  ] [trace] pfcp_session_look_up_pa
k_in_core 96 bytes, got far, far id AC2F68F0
2023-04-13T16:15:26.164681] [spgwu] [pfcp_sw  ] [trace] pfcp_session_look_up_pa
k_in_core 91 bytes, far id AC2F68B0
2023-04-13T16:15:26.164741] [spgwu] [pfcp_sw  ] [trace] pfcp_session_look_up_pa
k_in_core 91 bytes, got far, far id AC2F68F0
```

ubuntu@test-install-openair-cn: ~/openair-cn-cups/build/scripts

```
378
2023-04-13T16:15:18.054926] [spgwc] [pgwc_sx  ] [info ] handle_receive(16 bytes

2023-04-13T16:15:18.054989] [spgwc] [pgwc_sx  ] [trace] handle_receive_pfcp_msg
msg type 1 length 12
2023-04-13T16:15:18.055162] [spgwc] [pfcp     ] [trace] Sending HEARTBEAT_RESPO
SE, seq 5410490
2023-04-13T16:15:22.841781] [spgwc] [pgwc_sx  ] [info ] TIME-OUT event timer id
379
2023-04-13T16:15:22.842021] [spgwc] [pgwc_sx  ] [info ] PFCP HEARTBEAT PROCEDUR
 hash 1714894764 starting
2023-04-13T16:15:22.842343] [spgwc] [pfcp     ] [trace] Sending HEARTBEAT_REQUE
T, seq 13916161
2023-04-13T16:15:22.842988] [spgwc] [pgwc_sx  ] [info ] handle_receive(16 bytes

2023-04-13T16:15:22.843018] [spgwc] [pgwc_sx  ] [trace] handle_receive_pfcp_msg
msg type 2 length 12
2023-04-13T16:15:23.055686] [spgwc] [pgwc_sx  ] [info ] handle_receive(16 bytes

2023-04-13T16:15:23.055750] [spgwc] [pgwc_sx  ] [trace] handle_receive_pfcp_msg
msg type 1 length 12
2023-04-13T16:15:23.055855] [spgwc] [pfcp     ] [trace] Sending HEARTBEAT_RESPO
SE, seq 5410491
```

Fig. 4.3 5G Mobile Telecommunication Testbed: CN (Blue) and RAN (Pink)

Fig. 4.4 OpenCells SIM cards

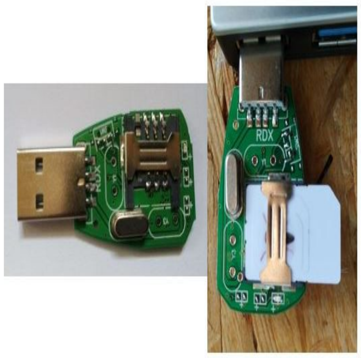


Fig. 4.5 Programming OpenCells SIM cards

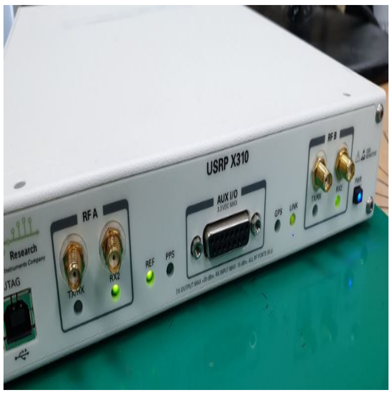


Fig. 4.6 USRP x310 with two UBX160 daughter boards

x310, [193]. An Intel 10GbE x520 NIC[3] was used to connect the USRP x310 to the eNB PC. An LTE-Softmodem on band 7 was used in order connect eNB to the USRP x310, which facilitates access towards the UEs.

- SDR USRP x310: The Universal Software Radio Peripheral (USRP) denotes radio equipment developed by Ettus Research. The employment of USRP allows the user to define and create a Software Defined Radio (SDR) connected to a host PC via a Gigabit Ethernet port. An Intel 10GbE x520 NIC connects the two Gigabit Ethernet ports between the eNB and the USRP device. Two UBX160 daughter boards were employed on USRP. For the purpose of this research radio frequency band 7 was employed.

- eNB: Evolved Node B was instantiated using the host PC under the Ubuntu distribution with a low latency Linux kernel. The software implementation of eNB is equipped with a scheduler that handles the Upper Link and Down Link of a PHY radio. As per the 3GPP specification, Frequency Division Duplex (FDD) and Physical Uplink Shared Channel (PUSCH) information is also available upon activation of the eNB.

- P4-BMv2 Switch [163] : Given the requirements for faster processing of packets, a BMv2 switch was installed between RAN-CN (on an NIC card). An ingress pipeline, a custom parser and an egress pipeline was developed working in parallel to perform match+action. The increased capacity for faster processing of packets established our motivation for their employment. As shown in our previous publication [165], UDP traffic has a greater throughput with a P4 implementation than with Open vSwitch

[3]Intel®Ethernet Converged Network Adapter x520 provides better flexibility and scalability for cloud and data centre environments

(OvS). Given that the communication between USRP and eNB are based on UDP traffic, the employment of a P4-BMv2 switch is an appropriate choice.

4.2.3 Core Network (CN)

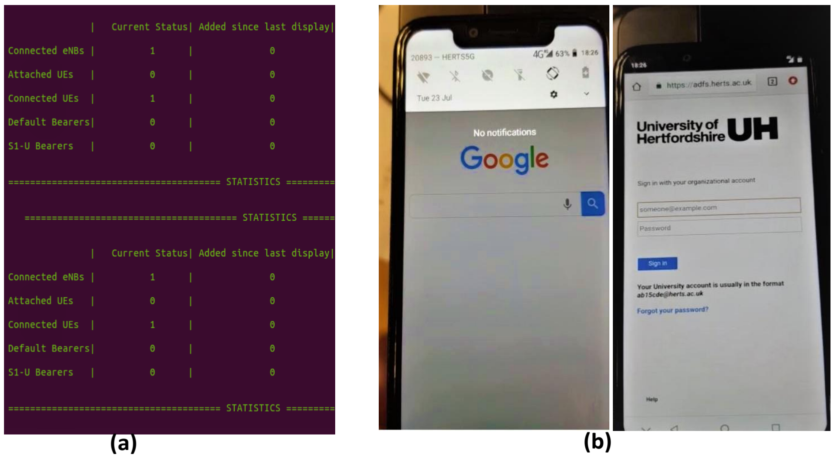
The CN is denoted in Fig.4.1 by the blue cloud. The following describe the various components of the CN, initialised on a Ubuntu UVT-Cloud [194] environment.

- HSS: HSS is a centralised database containing information relating to registered users and subscriptions. HSS provides data used for session creation, authentication and authorisation. The HSS database connects to the MME using a local loop-back. A Cassandra database stores and updates the records of the connected UEs.
- MME: The MME contains global information relating to the network. This includes attached and connected UEs, connected eNBs and bearers (default and S1U). MME communicates with eNBs, HSS and SPGW-C.
- SPGW-C: Provides control plane functions for the Serving Gateway (SGW-C) and Packet Data Network Gateway (PGW-C). It handles control requests from the MME and communicates with the SPGW-U. User traffic is tunnelled by SPGW-C as GTP (GPRS-Tunnelling Protocol).
- SPGW-U: This forwards user traffic between the Packet Data Network and the Internet. It is also, connected to the eNB.

4.2.4 Multi Access Edge Computing Platform (MEC)

The MEC deployment is depicted in yellow in Fig.4.1. For realising a 5G-MEC architecture four different types of deployments [2] are presented. We have employed (III) the MEC connected to a network edge point scenario due to its popularity [67, 195–197].

The MEC architecture was instantiated using two UVT-Cloud environments, the MEC Orchestrator and the MEC-UDP. Both MEC nodes have been virtualized as UVT-Cloud environments running Ubuntu Bionic servers. Each server has been instantiated with 2GB of RAM, 10GB of HDD and 1 CPU core. This is to reflect the lightweight, low processing power, that MEC nodes possess. An iPerf UDP server has been initiated on the MEC UDP node with the bind option, for the UE's to transfer UDP data.



```
               |   Current Status| Added since last display|
Connected eNBs |         1       |             0
Attached UEs   |         0       |             0
Connected UEs  |         1       |             0
Default Bearers|         0       |             0
S1-U Bearers   |         0       |             0

======================================= STATISTICS =========

   ======================================= STATISTICS ======

               |   Current Status| Added since last display|
Connected eNBs |         1       |             0
Attached UEs   |         0       |             0
Connected UEs  |         1       |             0
Default Bearers|         0       |             0
S1-U Bearers   |         0       |             0

======================================= STATISTICS =========
```

(a)
(b)
20893 – HERTS5G
Tue 23 Jul
No notifications
Google
https://adfs.herts.ac.uk
University of Hertfordshire UH
Sign in with your organizational account
someone@example.com
Password
Sign in
Your University account is usually in the format ab15cde@herts.ac.uk
Forgot your password?
Help

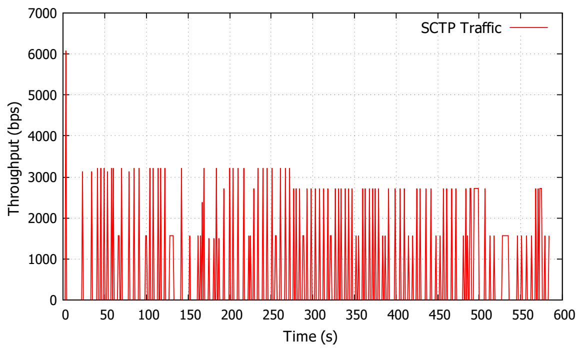
Fig. 4.7 Testing the Connection. (a) MME periodic update illustrating connected eNBs and UEs (b) Access of Google and University learning platform on a COTS UE.

4.3 Evaluation of the Testbed

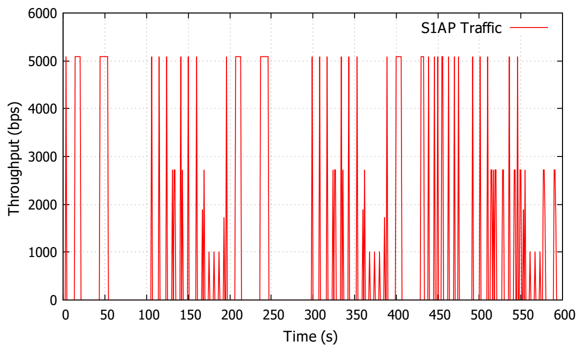
The traffic generation and workflow of the 5G testbed is shown in Fig. 4.1. Evaluation of the testbed is as follows. First we test the connectivity of the components in the testbed followed by the testbed with the UEs. Then, we evaluated the underlying 5G network traffic and finally we evaluated the user traffic in the developed testbed.

4.3.1 Testing the Connection

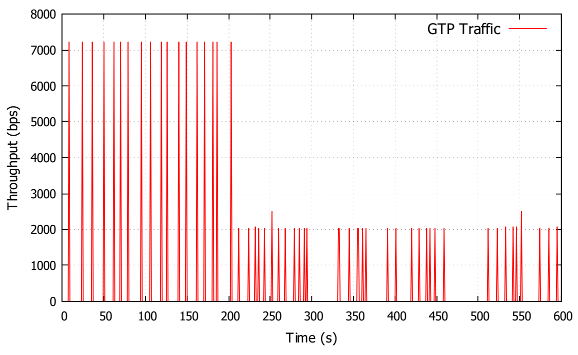
We first tested the UE connections. Fig.4.7(a) shows the periodic updates from the MME containing the connected devices of the network. Fig.4.7(b) shows the screenshots for the connection established after authentication and registration of the UEs to the testbed. The UEs were able to connect to the internet using the programmed mobile network- HERTS5G, which is an experimental 5G campus network. A ping message to the DNS also confirmed the successful connection in our testbed. We collected the underlying 5G network traffic once the UE connections had been established. Another key finding gathered from this experiment was that the P4-BMv2 switch was able to perform switching and packet processing for a higher volume of data (UDP traffic between USRP and RAN, see Fig 4.8f) with an average throughput of 2*10^9bps, without the switch collapsing.



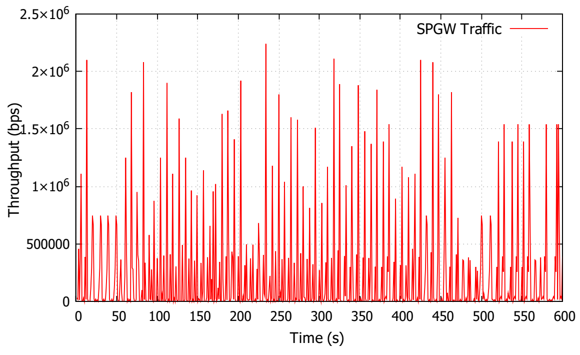
(a) SCTP Traffic



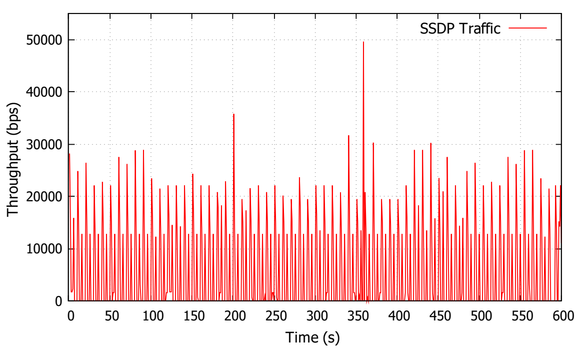
(b) S1AP Traffic



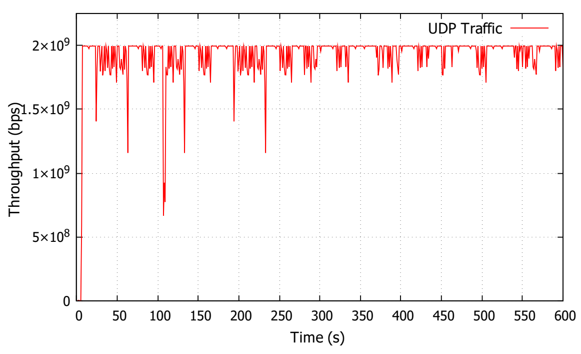
(c) GTP Traffic



(d) Periodic TCP traffic between eNB and MME

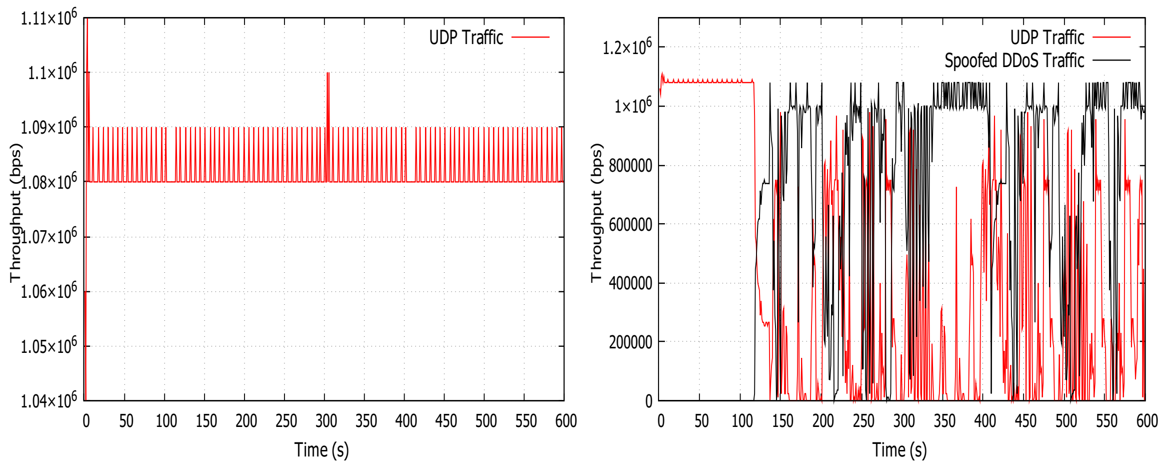


(e) SSDP traffic from SPGW-U towards DNS



(f) UDP data between eNB and USRP

Fig. 4.8 Underlying 5G Network Traffic



(a) UDP Traffic directed towards MEC node (b) UDP and Spoofed DDoS Traffic directed towards MEC node

Fig. 4.9 UDP and Spoofed DDoS Traffic directed towards MEC node

4.3.2 Underlying 5G Network Traffic

This traffic provides different types and protocols of traffic found in a 5G mobile telecommunication testbed. The underlying traffic (SCTP, S1AP, GTP, TCP (between MME and eNB), SSDP (between SPGW-U and DNS) and UDP (between USRP and eNB)) were observed in the collected dataset. Without introducing any user traffic, we collected the underlying traffic for a period of 600s. This type of traffic provides a dataset that can be used for training and testing algorithms used to enhance security for 5G communication.

Fig.4.8 presents six types of underlying 5G network traffic. SCTP traffic (Fig. 4.8a) between between SPGW-U, SGW-C and PGW-C occurs in the form of a heartbeat. Once generated, it is responded to with a heartbeat-acknowledgement. SCTP traffic carries the protocol number 132 and is an essential type of traffic in 5G mobile telecommunication for monitoring and detecting loss of sessions. S1AP traffic (Fig.4.8b) provides control plane signalling between RAN and EPC. S1AP traffic carries the protocol number 36412. The next type of traffic visible in a 5G mobile telecommunication network is GTP (Fig.4.8c) which is used by UEs to communicate with the DNS and other resources in the internet. GTP carries the protocol number 3386.

Fig.4.8d, represents the periodic TCP communication between MME and eNB. Since the MME require periodic updates about the connected/attached UEs, eNBs and bearers (tunnels for connecting UEs to packet data network) this periodic communication is extremely important for the network to maintain an accurate state. SSDP traffic (Fig.4.8e) aids the local network in service discovery. The captured traffic involves both SPGW-U and DNS. Fig.4.8f, represents the UDP traffic between the USRP and eNB. Due to the high volume and high speed required for communication between the two nodes, the communication is encapsulated as UDP traffic.

The above protocols and their underlying traffic have not been included in the following datasets (KDD Cup 99, NSL-KDD, CTU, UNSW NB-15 and InSDN). We note that the underlying TCP/UDP traffic in a 5G network is different in both volume and pattern from user-generated TCP/UDP traffic.

4.3.3 User Traffic

The third type of experiments generated legitimate UDP traffic from UE's to MEC node. A total of six iPerf clients were instantiated to communicate with the iPerf server at the MEC node. Data was collected for a period of 600s on each of the six hosts.

Fig.4.9a represents the average UDP throughput ($1.08*10^6$ bps) between the UE's and the MEC node. The fluctuations in the traffic are due to interference in the wireless medium. Underlying 5G network network traffic was also present but were not included in calculating the average UDP throughput.

4.3.4 Malicious Traffic

The fourth type of experiment generated malicious traffic from UE's towards MEC. A total of six clients initiated malicious traffic using hping3. Traffic such as DoS, Spoofed DDoS, botnet traffic, port scanning and TCP Syn Flood were employed in the malicious traffic category. Research by [101–104] highlighted that DoS, DDoS, and Spoofed DDoS with or without infected botnets have created a significant threat for MEC nodes. Data collated from their respective test beds [105] and case studies [106] revealed that prior to launching an attack adversaries employ scanning techniques either using TCP SYN messages or by conducting port scans to collect open port information. Research by [103] highlighted that due to low resources available in an MEC node, a traditional IDS which has a higher computational requirement may cause the IDS to malfunction, resulting in certain attacks being bypassed [107]. This also corroborates the findings of [4], on how traditional IDS are inadequate to be applied for 5G and beyond. Data was collected for a period of 600s on each of the six hosts. Underlying 5G network traffic and user traffic were also present during the malicious traffic transmission.

Fig.4.9b represents Spoofed DDoS traffic sent during the UDP transmission. The constant throughput of user traffic has been obstructed by the DDoS traffic. The throughput can be seen reaching 0bps at various intervals. Spoofed DDoS traffic was illustrated since it created a considerable service disruption compared to other types of malicious traffic.

4.4 Generating and Evaluating Datasets

4.4.1 5G Dataset

The collected traffic from the developed testbed was filtered using Python scripts for creating 5G datasets. Datasets included fields such as flow ID, source IP, source MAC, destination IP, source port, destination port, protocol, packet length, acknowledgment, and a binary label for classification. In our 5G dataset, depicted in Table 4.2, the underlying 5G network traffic and user traffic have been classified as legitimate and network attack traffic as malicious. Labeling network traffic into classes and the algorithms employed for same will shortly follow in the next chapter.

The quality of a dataset can be measured using the following indicators. Completeness, Uniqueness, Validity, Timeliness, Accuracy, and Consistency [54]. The 5G dataset that was accumulated using the testbed was aimed at achieving the aforementioned indicators.

1. Completeness: The dataset possesses zero incomplete values. As the network data was accumulated in real-time, a Python function was programmed in order to remove incomplete rows of data. This function was particularly useful in avoiding biased analysis. Furthermore, the NeT2I algorithm was also programmed to read a complete set of columns in a row, in order to generate an image. These strategies were taken in order to ensure the completeness of the dataset.

2. Uniqueness: The number of unique rows in the 5G dataset is shown in Table 4.2. Out of the collected traffic, we filtered and refined the traffic to a total of 1,865,935 rows to remove redundant, repetitive, and empty rows of data. We employed several Python scripts for the refinement and creation of the datasets.

3. Validity: Authors in [55] measured the validity of the dataset using the quality of the metadata. In the 5G dataset, in addition to the collated network traffic, additional features were generated that aided algorithms in detection. These additional features can be classified as metadata of the dataset.

4. Timeliness: The publicly available datasets (UNSW NB-15 and InSDN), to which the generated dataset was compared, were collected in the years 2015 and 2020. The change in traffic patterns, protocols, and volume differ significantly in the generated dataset as opposed to the publicly available datasets. This may contribute towards a higher false positive rate to the intrusion detection problem in 5G.

5. Accuracy: [56] measured the accuracy determining inaccurate values using functional dependence rules. In the generated dataset, each column of collected network traffic

only carried data that accurately represented the column title. For example, the column which included data on MAC addresses only carried MAC addresses throughout the dataset. The dataset was organised by using a shell function. Similarly, additional features were calculated based on the collected network traffic, and these additional features were assigned a value based on binary classification.

6. Consistency: As per [57], data consistency can be mentioned as the evaluation of data from two different datasets. The 5G dataset has been evaluated using two other datasets. They are the UNSW NB15 and InSDN datasets. The collated features can also be found in the aforementioned two datasets.

4.4.2 UNSW NB-15 and InSDN Datasets

Table 4.3 and Table 4.4 present a summary of the UNSW NB-15 and InSDN datasets which we employed for detection and evaluation along with our 5G dataset. From Tables 4.2, 4.3 and 4.4, we note that the underlying 5G network traffic is absent from the UNSW NB-15 and InSDN dataset. Xu *et al.* [137] highlight that as a result of an imbalanced or partial traffic classification, a higher False Positive rate follows. Although augmentation schemes have been employed to mitigate this drawback, Xu *et al.* state that the correct representation of generated datasets provides better results with a negligible False positive rate. Hence, it is important to be able to classify underlying 5G network traffic as legitimate traffic. The total number of flows in both datasets (Table 4.2, Table 4.3, and Table 4.4) can be classified into two classes, malicious and legitimate. Hence, we employed binary classification in our detection methodology (CNN), which will be presented in the following chapter.

4.5 Chapter Summary

In this chapter, we developed a 5G mobile telecommunication testbed to produce 5G datasets that can be used to study 5G traffic malicious attacks and their characteristics. We conducted this research to advance the effectiveness of security based research for 5G and beyond, using 5G datasets instead of simulators or emulators. Our UDP server has been instantiated using iPerf but the testbed is otherwise free from simulation or emulation. We collected traffic unique to a 5G mobile network, conducted malicious attacks on an MEC node, studied the service disruption and presented the associated data for the above. The employment of the testbed for this research paved the following findings which were unexpected. The underlying 5G network traffic, traffic between RAN and USRP, and protocol types. User traffic and malicious traffic are commonly found in any publicly available dataset. However,

Traffic Type		Number of Flows
Underlying 5G Network Traffic	SCTP	150,706
	S1AP	100,472
	GTP Traffic	50,236
	eNB and MME Traffic	251,176
	SSDP Traffic	25,118
	eNB-USRP UDP Traffic	401,882
	Total	979,590
User Traffic	UDP Traffic	468,847
Malicious Traffic	DoS	14,976
	DDoS	39,700
	TCP Syn Flood	142,604
	Botnet Traffic	39,700
	Port Scanning	180,518
	Total	886,345
Total Flows		1,865,935

Table 4.2 Number of unique rows in the collected 5G dataset

UNSW NB-15		Number of Flows
User Traffic	TCP	1,492,153
	UDP	990,144
	ICMP	524
	Other	524
	Total	2,218,755
Malicious Traffic	Fuzzers	24,246
	Reconnaissance	13,987
	Shellcode	1511
	Analysis	2677
	DoS	16,353
	Exploits	44,525
	Worms	174
	Generic	215,481
	Total	321,283
Total Flows		2,540,038

Table 4.3 Description of UNSW NB-15 dataset

InSDN		Number of Flows
User Traffic	TCP	34,115
	UDP	33,920
	Other	389
	Total	68,424
Malicious Traffic	DDoS	121,942
	Probe	98,129
	DoS	53,616
	brute-force-attack	1405
	Exploit	17
	Web attack	192
	Botnet	164
	Total	275,515
Total Flows		343,939

Table 4.4 Description of InSDN dataset

the presence of underlying network traffic which aids in maintaining the state of a mobile telecommunication network was not anticipated. Also, network traffic pattern, the volume, and the frequency between RAN and USRP device was not anticipated. Lastly, protocols such as GTP and S1AP are unique to a mobile telecommunication network. The presence of these protocols would not have been included in the generated dataset, had the testbed not been employed. The aforementioned can be summarised as findings that were generated due to the employment of a testbed. Details regarding the CNN detection algorithm, and encoding/decoding algorithm employed in the RGB image generation together with a discussion on the computational complexity of the algorithms employed will follow in the next chapter. We also test our algorithms on two publicly available datasets and the collected 5G dataset, where we aim to answer the research question: How can we design new algorithms for intrusion detection using a DL agent in 5G-MEC?

Chapter 5

New Algorithms for the Detection of Malicious Traffic in 5G-MEC

This chapter presents a new Intrusion Detection System using a 3-layer Convolutional Neural Network (CNN), capable of identifying malicious network traffic. We employ a new injective algorithm to encode network traffic without loss of information. We also include a new algorithm to decode, and encoded RGB images back into network traffic. We evaluate the proposed Intrusion Detection System (IDS) in terms of its computational complexity for example time, memory and CPU utilisation for the encoding and decoding algorithms, and its accuracy and loss during training and detection. Lastly, we compare the proposed IDS against a significant IDS algorithm that uses a different approach for encoding, decoding and CNN detection.

5.1 Introduction

For securing 5G networks and Multi-access Edge Computing (MEC) infrastructure, authors at [59] and [60] claimed that the application of Deep Learning (DL) to the problem of detecting malicious traffic would be a better approach than using a Machine Learning (ML) approach, leading to greater accuracy due to the complexity arising from communication between components and users in various 5G (and beyond) use cases [114]. Deep Learning ([132] and [198]), has caught the attention of both academia and industry, with applications of DL increasing, following the application of deep convolutional networks in computer vision. The ease of training and generalisation, in comparison to other fully connected networks together with the high accuracy rate achieved with Convolutional Neural Networks (CNN) [132], has prompted us to explore CNN as a viable option for the detection of malicious traffic in the 5G network infrastructure.

One of the greatest challenges in the application of CNN to the study of network security involves the process of encoding data into a form recognisable by the CNN. As CNNs accept images for training and testing, data collected from a dataset has to be converted into images. Existing research uses various methods to encode network traffic, data into images that can be used to train a CNN algorithm.

Research published in [134], [144], [145], [146], and [147] employed grayscale images to represent and encode their desired features of network traffic for training and testing a CNN algorithm. However, representing network traffic by grayscale images can lead to a loss of information since modern network traffic due to heterogeneity and complexity can contain data that exceeds a pixel value in the grayscale (0-255).

Due to this limitation on grayscale images, authors in [1] and [140] employed mechanisms to encode network traffic as RGB images. Improvements in accuracy were observed [140], when employing RGB images in preference to opposed to grayscale images. Authors of [1] and [140], employed a tiled image approach along the x and y axis, for a given pixel length and width. Although, these images are capable of producing a higher accuracy than grayscale images, RGB can represent a pixel value between 0 - 16,777,215, leading to in a tiled image that places a high demand on CPU, memory and time of execution.

Since the MEC infrastructure will possess low processing power, a CNN based IDS must produce images for the detection algorithm requiring less computation. To the best of our knowledge, research based in encoding network traffic into images, has not discussed computational complexity in the application of their respective algorithms.

In this chapter, we propose a new method in the encoding and decoding process. We have evaluated the computational complexity that arises with the use of our proposed encoding and decoding mechanism and compared this against a similar encoding mechanism in RGB that uses a tiled image approach. The conducted experiments in this chapter have been presented in Table 5.1.

The key contributions of this chapter include:

- A new algorithm to encode network traffic for example, IP addresses, MAC addresses to RGB Images and a new algorithm to decode, encoded RGB images into network traffic.
- A new IDS using CNN and the proposed encoding and decoding algorithms for the detection of malicious network traffic.

Experiments
RQ3: New Algorithms for the Detection of Malicious Network Traffic
1. Study of the existing Machine Learning and Deep Learning Algorithms
2. Study of the existing methods of data encoding
3. Implement algorithms for data encoding and decoding
4. Implement a detection algorithm by employing CNN
5. Evaluate existing publicly available datasets
6. Encode data from the Datasets using the developed encoding algorithm
7. Evaluate the computational complexity of the encoding algorithm against literature
8. Train and Test the CNN algorithm
9. Apply the Confusion Matrix

Table 5.1 Experiments conducted in this Chapter

- Evaluation of the proposed IDS in terms of computational complexity in, for example, time, memory and CPU utilisation, together with accuracy and loss in training, validation and detection.
- Comparison of the proposed IDS against a significant IDS that uses a different approach for encoding and CNN detection.

5.2 Proposed Algorithms

In this section, we present new algorithms for encoding network traffic as RGB images and for decoding the images back to network traffic. We also present the CNN detection algorithm that we have employed for identifying traffic as either malicious or non-malicious

5.2.1 Encoding Network Traffic to Images (NeT2I Algorithm)

The NeT2I algorithm describes the pseudo-code used for the Python script utilised in creating PNG images from network traffic that has been saved as a CSV file. The NeT2I algorithm outlines two primary functions. They are, encode network data collated in a CSV file into RGB values and subsequently translate those RGB values into an image in the PNG format. `def encodeCSVToRGB (Input):` This function takes a CSV file as an input. The function initialises a `2D Tuple` to store RGB values generated through the function. The function iterates through each line in the CSV, storing values in a new `Tuple` for each line. This loop also determines the variable type in each value in each line. For integer values less than 2^{24}, such as flow ID, port numbers, protocol number and packet length, a sub-function `convertIntToRGB (value)` maps them to RGB values and saves to the current `Tuple`. Since IP addresses and MAC addresses are larger than 24 bits, mapping them without losing the integrity of data is a challenge. In the NeT2I algorithm, we adopted a delimited approach and present a detailed description of this process below.

MAC address

Mapping of MAC addresses were only carried out to the proposed 5G dataset. Mapping a 48-bit MAC address into an RGB value that can be used to create a PNG image is a challenge due to the 24-bit space of the RGB scheme. We used the colon symbol (:), which divides the MAC address into six octets (NeT2I Algorithm: line 9). By employing a sub-function `convertMACToRGB (value)` each Hexadecimal value was mapped to its corresponding integer value using another sub-function `HexToInt(hex)`. This provided us with six integer values that can be used to map the address to its respective RGB values. The resulting values were appended to the `Tuple`. These values were then used in the image creation process.

IP address

We employed the IPv4 addresses available in the UNSW NB-15 and InSDN datasets. Mapping a 32-bit IP address into an RGB value that can be used to create a PNG image is a challenge due to the 24-bit space of the RGB scheme. We used the decimal point (.), which divides the IPv4 address into four octets (NeT2I Algorithm: line 13). This provided us with four integer values that can be used to map the address to its respective RGB values. This was carried out by employing the `convertIPToRGB (value)`. The resulting values were appended to the `Tuple`. If the value does not fit any of these types, an error is flagged, indicating that the value cannot be read. Finally, the function concludes by saving the RGB `Tuple` to the overarching `2D Tuple`, consolidating the encoded RGB information. The values in the `2D Tuple` were then used to create a PNG image.

Mapping RGB values and creating PNG images

Upon successfully mapping integer values to RGB values, we saved them in a Python `2D Tuple`. Since RGB is a representation of multiple values that require storing in the same element, the use of the Python data structure `2D Tuple` was ideal. The `def RGB_to_PNG (2DTuple)` in line 19 takes the `2D Tuple` input where all the RGB values have been saved. The function iterates through each tuple in the `2D Tuple` and within, each tuple, iterates through the individual elements which store RGB values. For each value, we employed a condition check to identify stored values larger than 2^{24}. If the values meet the conditional check, they are mapped to a PNG file by employing a sub-function **`savePNG`** which invokes another sub-function **`createPNG (values)`** where individual RGB values were passed as input. We utilised the PIL (Pillow) library within **`createPNG`** function. Finally, the algorithm returns the output PNG, providing the result of the RGB to PNG conversion process.

Encoding Algorithm 1: NeT2I

```
Input: input.csv
Output: output.png
def encodeCSVToRGB (Input):
    2DTuple = [[ ]]
    for line in csvfile do
        TUPLE = [ ]
        for value in line do
            if type(value) == int then
                rgb = convertIntToRGB(value)
                saveRGBValueToTuple(rgb, TUPLE)
            if type(value) == MAC then
                value = HexToInt(hex)
                rgb= convertMACToRGB(value)
                saveRGBValueToTuple(rgb, TUPLE)
            if type(value) == IP then
                rgb= convertIPToRGB(value)
                saveRGBValueToTuple(rgb, TUPLE)
            else
                Error /* Can't read value */
            saveRGBTupleTo2DTuple(TUPLE, 2DTuple)
def RGB_to_PNG (2DTuple):
    for tuple in 2DTuple do
        for value in tuple do
            if value > $2^{24}$ then
                Error /* Can't draw value to RGB */
            Output = savePNG(createPNG(value))
        return Output
```

Decoding Algorithm 2: I2NeT

```
Input: input.png
Output: output.csv
def Images_to_CSV (Input):
    2DTupple = [[ ]]
    for png in PNGs do
        Tuple = [ ]
        int = convertPNGToInteger(png)
        saveIntegerToTuple(Tuple)
        convertIntegerToMAC (values 1_2_3_4 _5 _6):
            for value in values 1_2_3_4_5 _6 do
                MAC += value
                MAC += ':'
            return MAC
        convertIntegerToIP (values 1_2_3_4):
            for value in values 1_2_3_4 do
                IP += value
                IP += '.'
            return IP
        createSourceMACFromInt (tuple):
            /* element(1, 2, 3, 4, 5, 6)                          */
            SourceMAC = convertIntegerToMAC(tuple)
            return SourceMAC
        createSourceIPFromInt (tuple):
            /* element(7, 8, 9, 10)                               */
            SourceIP = convertIntegerToIP(tuple)
            return SourceIP
        createDestinationIPFromInt (tuple):
            /* element(11, 12, 13, 14)                            */
            DestinationIP = convertIntegerToIP(tuple)
            return DestinationIP
        return 2DTuple
        saveIntTupleTo2DTuple(Tuple, 2DTuple)
    Write_2DTuple_to_CSV ():
        WriteToCSV (2DTuple, Output)
    return Output
```

Detection Algorithm 3: CNN3L

```
Input: PNGs
Segregate Dataset (Images):
    while Upload Images do
        if $Images \equiv Corrupt$ then
            Remove or Discard corrupt images
        else
            Split 80% as train images and 20% for test
Create CNN Model ():
    Input RGB Image 150*150 pixels
    Convolutional Layer 1 (32, (3,3))
    MaxPooling (2,2)
    Convolutional Layer 2 (64, (3,3))
    MaxPooling (2,2)
    Convolutional Layer 3 (128, (3,3))
    MaxPooling (2,2)
    while load train images do
        for Randomly selected images do
            Run Image Augmentation and Replace
    while $Steps \neq NumberOfEpochs$ do
        for Load images by BatchSize do
            Train and Test CNN
            Save to Array1
        Steps++ if $Steps \equiv NumberOfEpochs$ then
            Calculate Average Values of Array1
            Print Plots
        else
            Print Error /* Array Empty */
Detection ():
    $LoadImagesfromNeT2I\_Algorithm$ classes = model.predict(images)
    if $classes > 0.5$ then
        print Legitimate Traffic
    else
        print Malicious Traffic
```

5.2.2 Decoding Images to Network Traffic (I2NeT Algorithm)

The I2NeT algorithm describes in pseudo code the steps involved in generating network traffic from PNG images. The algorithm can only decode PNG images that were generated using the NeT2I algorithm since the underpinned algorithm functions by mapping RGB values found in an image to corresponding fields for network traffic.

The function `def Images_to_CSV (Input):` takes the PNG images as the input. The function initialises a `2D Tuple` to store processed data. For each PNG that is received as an input a `Tuple` is created. The pixel values of the PNG are converted to integers by employing the sub-function `convertPNGToInteger (PNG):`. The resulting integer values are saved to the `Tuple`, and then appended to the `2D Tuple`. The images were created by encoding string fields such as MAC addresses, and IP addresses, these values were treated differently to other integer values, during the decoding stage.

Decoding MAC address

Similarly, a set of acquired integer values, from the previous mapping stage correspond to the sender's MAC address. Stored integers are converted to Hexadecimal values and stored in the same element. The function `convertIntegerToMAC (values 1_2_3_4_5_6):` conducts the aforementioned task. Since the converted Hexadecimal values are a representation of an octet in the MAC address, we use a delimited approach(:) for appending octets into a MAC address. We only converted the source MAC addresses, as this will aid us in detecting Spoofed IP addresses. The employed function can be found in the I2NeT algorithm: line 17. This function (`convertIntegerToMAC (values 1_2_3_4_5_6):`) has been called as a sub-function in `createSourceMACFromInt (tuple):` which returns the SourceMAC address.

Decoding IP address

Once the integer value has been derived from the mapping stage, corresponding array elements of the sender and receiver IP addresses are decoded separately. Since the converted integer numbers are a representation of an octet in the IP address, we use a delimited approach for appending octets into an IP address. The function `convertIntegerToIP (values 1_2_3_4):` was aimed at conducting the aforementioned tasks. In our algorithm, we use two separate functions `createSourceIPFromInt (tuple):` which utilised elements 7, 8, 9, 10 as inputs and `createDestinationIPFromInt (tuple):` which utilised elements 11, 12, 13, 14 as inputs to return the IP addresses corresponding to the source and the destination. The employed function for mapping RGB values to source IP address and destination IP address can be found in the I2NeT algorithm: lines 20 and 23, respectively. The returned values from the functions were saved to a `2D Tuple` from the `Tuple` by calling the function `saveIntTupleTo2DTuple(Tuple, 2DTuple).`

Creating the initial network traffic

After successful mapping of RGB values (in a Tuple) to an Array, we then write the contents of this array to a CSV file. The pseudo-code instructions can be found in the I2NeT algorithm: line 28, outlining the function `Write_2DTuple_to_CSV ():`. Upon writing the last element in the array, we save each element separated by a delimiter (‘,’) to ensure that the file corresponds to a CSV file.

5.2.3 Detection Algorithm

The CNN3L algorithm, conducts a binary classification of network traffic into non-malicious and malicious traffic, using `TensorFlow` and `Keras` libraries on Google Colaboratory.

Prior to building the model, we initially segregated the dataset using the `Segregate Dataset(Images):` function. This function iterates through the images created using the `NeT2I` algorithm. If an image is identified as corrupt, it is removed or discarded. Otherwise, the dataset is split into an 80% training set and a 20% test set. In line 7 of the CNN3L algorithm, the pseudo block for the `CNN Model():` has been presented. This model is designed to take RGB images of size 150 x 150 pixels as input. It consists of three layers of Convolutional 2D with corresponding MaxPooling layers, progressively extracting features from the images. We employed three convolutional layers as opposed to one or two due to the increase in accuracy.

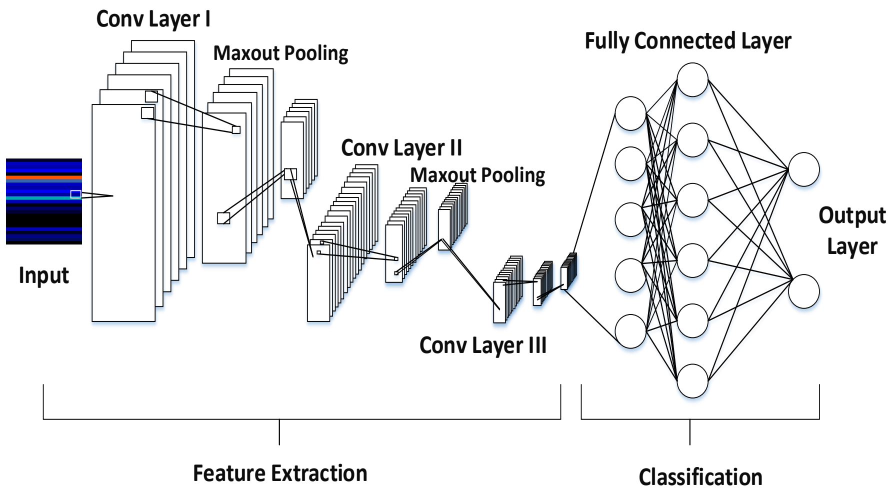


Fig. 5.1 Convolutional Neural Network

The application of three convolutional layers with the higher accuracy found in the detection of malicious traffic has also been discussed in [140]. Similarly, the authors [138, 139, 155] provided evidence for same, when the CNN layers fall under three or surpass three, the accuracy reduces with a higher loss. We employed the sigmoid activation function since a binary classifier can be represented using a one neuron, setting the value 0 for non-malicious and 1 for malicious traffic. The computational complexity of the CNN3L algorithm was not considered for this research. This is due to CNN3L being launched in the Google Colaboratory environment. However, the work by [135] presented the resource utilisation and energy consumption of various CNN algorithms on smartphones. All of the presented algorithms have a higher number of convolutional layers and fully connected layers as opposed to the proposed CNN3L algorithm. Similarly, authors at [136], evaluated CNN algorithms in edge devices by employing an algorithm with three convolutional layers and two fully connected layers, similar to that of the CNN3L algorithm, due to low resource utilisation and high accuracy. Fig. 5.1 illustrates the proposed CNN3L algorithm using the parameters highlighted in Table 5.8.

Upon model creation, pseudo-randomly selected images in the training set were sent and replaced through an image augmentation function, in order to prepare our model to detect data or attributes that were not in the original dataset. This is an important function to consider since the heterogeneity of 5G mobile telecommunication traffic can create network traffic patterns that the training set may not possess.

CNN3L, then enters the training loop when `while Steps≠ Number Of Epochs:`, where it loads the training images to the CNN model that has gone through image augmentation, and trains the model. The training process occurs in batches, updating model parameters iteratively. The resulting performance metrics are saved to `Array1`. After completing the set number of epochs, the average value of the performance metrics in `Array1` are calculated. In the CNN3L algorithm, we considered different epochs and batch sizes, in order to achieve the highest accuracy (line 18). These were incremented with each iteration once the optimal batch size was determined (line 19).

Lastly, the algorithm transitions to the detection phase by calling the `Detection():` function. (See line:28) We utilised the `model.predict(images):` function, where the images from NeT2I served as an input. If the `classes` exceed a threshold of `0.5`, the image is classified as legitimate or non_malicious. Otherwise, it is classified as malicious.

5.3 Evaluation Metrics for the New Algorithms

5.3.1 Workflow and Dataset

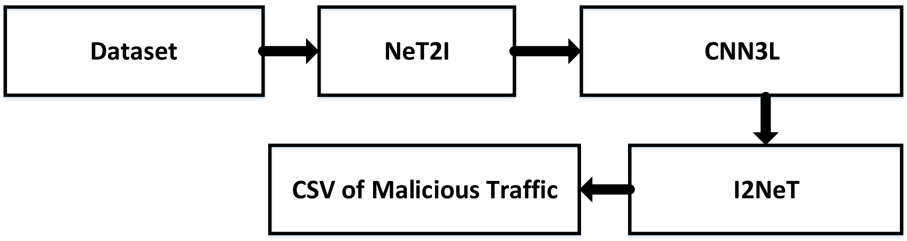


Fig. 5.2 Workflow of the Proposed Algorithms

5G Dataset

Table 5.2 described the features collected from 5G-MEC mobile telecommunication testbed. The traffic was initially collected in the form of a pcapng file, which was filtered into creating datasets. For the features mentioned above, a label was included to classify traffic which resulted in a dataset of 12 features. Details pertaining to Source port variation and destination port variation will follow in the following chapter.

5G Dataset Feature	Description
f1	Flow ID
f2	Source IP
f3	Source MAC
f4	Destination IP
f5	Source Port
f6	Destination Port
f7	Protocol
f8	Packet size
f9	Source Port Variation
f10	Destination Port Variation
f11	Flags

Table 5.2 Features Selected from the 5G Dataset

UNSW NB 15 Dataset

Amongst the publicly available datasets, the UNSW-NB15 dataset contains the most recently collected data for malicious and non-malicious traffic. The collected data has been represented using 49 features with 12% of the available traffic in the UNSW-NB15 dataset corresponding to malicious traffic. We randomly collected a total of 12500 lines from each traffic class (malicious and non-malicious) and created a subset of the dataset with 25000 rows. The resulting CSV file was further filtered to reflect the features employed in [1] for compatibility purposes in our research. A final dataset with 11 features (f1, f2, f3, f4, f5, f7, f8, f15, f17, f23 and f42) resulted together with a label for traffic classification, generating a CSV file containing 12 features. The features are described in Table 5.3. The workflow of the proposed algorithms and their applications on the UNSW NB-15 dataset is shown in Fig. 5.2. The CSV file extrapolated from the UNSW NB-15 dataset was used as input for the NeT2I algorithm for generating PNG images based on network traffic. Generated PNG images were used as input for the CNN3L algorithm. Following detection by the CNN3L algorithm, the I2NeT algorithm was used to decode the images back to a CSV file in order to test the accuracy of our detection algorithm.

InSDN Dataset

Another dataset that we considered was InSDN dataset that was collected recently by researchers at University College Dublin [53]. The dataset has a collection of 83 features with 80% of traffic corresponding to malicious traffic. We randomly collected a total of 12500 lines from each traffic class (malicious and non-malicious) and created a subset of the dataset with 25000 rows. The resulting CSV file reflects the features employed in [1]. The dataset contains 11 features (f2, f3, f4, f5, f6, f10, f12, f14, f18, f29 and f51) along with a label for traffic classification. The features are described in Table 5.4. The same workflow mentioned in Fig. 5.2 has been employed for InSDN dataset as well. The extrapolated CSV was used as input for the NeT2I algorithm for generating PNG images based on network traffic. Generated images were used as input for the CNN3L algorithm. Upon detection by the CNN3L algorithm, the I2NeT algorithm was used to decode the images back to a CSV file in order to test the accuracy of our detection algorithm.

UNSW NB-15 Feature	Description
f1	Source IP
f2	Source Port
f3	Destination IP
f4	Destination Port
f5	Protocol
f7	Duration
f8	Source to Destination bytes
f15	Source bits per second
f17	Source to Destination packet count
f23	Mean packet size transmitted by the source
f42	No. of connections that contain the same service and destination address

Table 5.3 Features Selected from the UNSW-NB15 Dataset

InSDN Feature	Description
f2	Source IP
f3	Source Port
f4	Destination IP
f5	Destination Port
f6	Protocol
f18	Flow Duration
f10	Total packets in forward direction
f12	Total size of the packet in forward direction
f14	Mean size of the packet in forward direction
f49	Number of Flow Bytes per second
f51	Number of forward packets

Table 5.4 Features Selected from the InSDN Dataset

5.3.2 Evaluation Metrics for the NeT2I and I2NeT Algorithms

For evaluating the encoding algorithms, we compared the accuracy and computational complexity of each in terms of time of execution, CPU and RAM. An algorithm that can encode network traffic as images and decode images back to network traffic with low computational complexity and high accuracy will be advantageous for the detection of malicious traffic in 5G-MEC networks.

Time for execution

For measuring the time of execution, we employed the Python Time library. As discussed in [199], identifying the accurate execution time of a program is important being preferable to using an estimate for the execution time. Variation in execution time can affect performance and efficiency, particularly important when applying the algorithm to an intrusion detection problem based in an environment with low resources.

CPU Utilisation

For CPU utilisation we employed the Python psutil library in order to collect the percentage use for the algorithm. Our code uses the standard single-threaded Python execution with global interpreter lock. Hence, our algorithms can only be executed on one single CPU thread at a time. Since our algorithms are expected to be executed at an MEC node with low processing capabilities, it is crucial that our algorithms effectively utilise CPU resources for an optimal window of time.

Memory Utilisation

As discussed in [200], memory allocation is not as precise as the CPU allocation or usage. In order to determine memory allocation for our algorithms, we employed the Python library memory_profiler. Computer systems tend to over-allocate memory to a process for efficiency of execution, and as garbage collection doesn't occur instantaneously, an average value for the memory allocation is used to compare the performance.

5.3.3 Evaluation Metrics for the CNN3L Detection Algorithm

We evaluated our CNN3L algorithm based on configurations, for example, convolutional layers, epochs and batch sizes, in order to understand the percentage deviation that occurs in the accuracy of the detection algorithm when different configurations are employed.

We considered evaluating our CNN3L algorithm in terms of overfitting. This occurs when the CNN model perform with significant results for the training data and can not perform well for the validation data. We evaluated overfitting by observing and calculating the difference between training and validation lines for both accuracy and loss.

The following metrics [5] were used in terms of accuracy (A), precision (P), recall (R) and F1-Score (F1) to evaluate the proposed CNN detection algorithm.

$$A = \frac{TP+TN}{TP+FP+TN+FN} \tag{5.1}$$

$$P = \frac{TP}{TP+FP} \tag{5.2}$$

$$R = \frac{TP}{TP+FN} \tag{5.3}$$

$$F1 = \frac{P*R*2}{P+R} \tag{5.4}$$

TP stands for the number of positively predicted attacks, FP stands for negatively predicted attacks on non-malicious traffic, TN represents non-malicious traffic that was correctly predicted as normal, and finally, FN stands for malicious traffic that was predicted as normal.

5.3.4 Existing Algorithms for Comparison

To evaluate the performance of the algorithms NeT2I, I2NeT, and CNN3L, we would like to compare our IDS with an existing IDS. The IDS presented in [1], is a recently published study is selected because of its use of 2D RGB images for network traffic, with the application of CNN for the detection of malicious traffic and for its high accuracy. We evaluated our algorithms in terms of the performance metrics discussed in Section 5.3.2 against [1]. The algorithms used in comparing the two IDs are as follows:

- NeT2I is compared with the encoding algorithm used in [1]. Both transfer traffic data to images for the detection of malicious traffic. The NeT2I algorithm produces one dimensional RGB images while the encoding algorithm used in [1] generates tiled RGB images.

- The CNN3L algorithm is used for both IDSs. Although a two layer CNN is used in [1], due to the recorded performance gain in employing three convolutional layers as opposed to two layers [140], we employed, for comparative purposes CNN3L for both IDSs.

- The I2NeT algorithm was evaluated against the decoding algorithm of [1], although not explicitly given in their work. This algorithm is capable of generating a CSV file for the

Algorithm	Traffic Class	Number of Images	Total Execution Time	Execution Time Per Image	CPU Utilisation	CPU Utilisation Per Image	Memory Utilisation
NeT2I	Malicious	12500	10.1s	0.000808	100 %	0.008%	20.2%
	Non-Malicious	12500	10s	0.0008	100 %	0.008%	20.1%
Encoding Algorithm used in [1]	Malicious	12500	28.2s	0.002256	100 %	0.008%	27%
	Non-Malicious	12500	28s	0.00224	100 %	0.008%	27%
I2NeT	Malicious	12500	6s	0.00048	100 %	0.008%	21%
	Non-Malicious	12500	6s	0.00048	100 %	0.008%	21%
Decoding Algorithm used in [1]	Malicious	12500	28s	0.00224	100 %	0.008%	28%
	Non-Malicious	12500	28s	0.00224	100 %	0.008%	28%

Table 5.5 Computational complexity of NeT2I and I2Net against the encoding and decoding algorithms in [1] for 5G Dataset

Algorithm	Traffic Class	Number of Images	Total Execution Time	Execution Time Per Image	CPU Utilisation	CPU Utilisation Per Image	Memory Utilisation
NeT2I	Malicious	12500	6s	0.00048	100%	0.008	19%
	Non-Malicious	12500	6s	0.00048	100%	0.008	19%
Encoding Algorithm used in [1]	Malicious	12500	27s	0.00216	100%	0.008	28%
	Non-Malicious	12500	27s	0.00216	100%	0.008	28%
I2NeT	Malicious	12500	6s	0.00048	100%	0.008	20%
	Non-Malicious	12500	7s	0.00056	100%	0.008	20%
Decoding Algorithm used in [1]	Malicious	12500	28s	0.00056	100%	0.008	29%
	Non-Malicious	12500	29s	0.00232	100%	0.008	28%

Table 5.6 Computational complexity of NeT2I and I2Net against the encoding and decoding algorithms in [1] for UNSW NB-15 Dataset

associated network traffic from a collection of tiled PNG images by reading the pixel value of each tile in the image and mapping the value to its respective integer value. Since the encoding algorithm of [1] masked the IP address of source and destination, a simple RGB to Integer function was employed to decode all tiles in the image into a corresponding a CSV file.

5.4 Results and Analysis

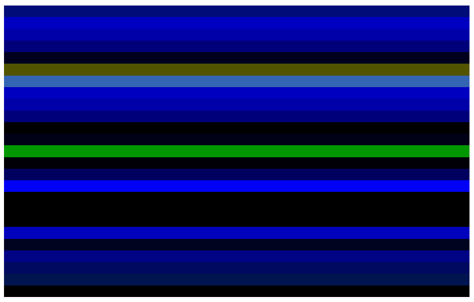
We collected the data on a UVT_Cloud deployment running Ubuntu 18.04 LTS, with a single CPU, 8GB of RAM and 20 GB of HDD space. 25000 images representing two different network classes were created consisting of 12500 images for malicious traffic and 12500 for non-malicious traffic. The images were subsequently grouped in the ratio 80:20, for training

Algorithm	Traffic Class	Number of Images	Total Execution Time	Execution Time per Image	CPU Utilisation	CPU Utilisation per Image	Memory Utilisation
NeT2I	Malicious	12500	10s	0.0008	100%	0.008%	20.8%
	Non - Malicious	12500	10s	0.0008	100%	0.008%	20.8%
Encoding Algorithm used in [1]	Malicious	12500	30s	0.0024	100%	0.008%	29.4%
	Non - Malicious	12500	29s	0.00232	100%	0.008%	29.4%
I2NeT	Malicious	12500	6s	0.00048	100%	0.008%	20%
	Non - Malicious	12500	6s	0.00048	100%	0.008%	20%
Decoding Algorithm used in [1]	Malicious	12500	30s	0.0024	100%	0.008%	29.8%
	Non - Malicious	12500	29s	0.00232	100%	0.008%	29.8%

Table 5.7 Computational complexity of NeT2I and I2Net against the encoding and decoding algorithms in [1] for InSDN Dataset

and testing respectively. Randomly selected images were passed to the image augmentation process to eliminate overfitting in the algorithm. The images were processed by the CNN3L algorithm in batches in order to increase efficiency for the experiment.

5.4.1 Encoded Images


(a) An image from the NeT2I algorithm

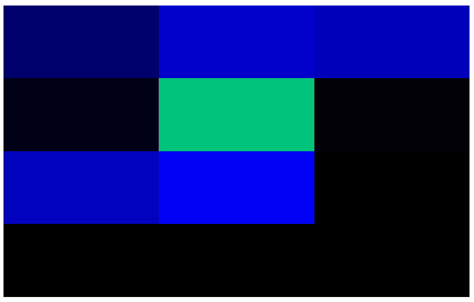
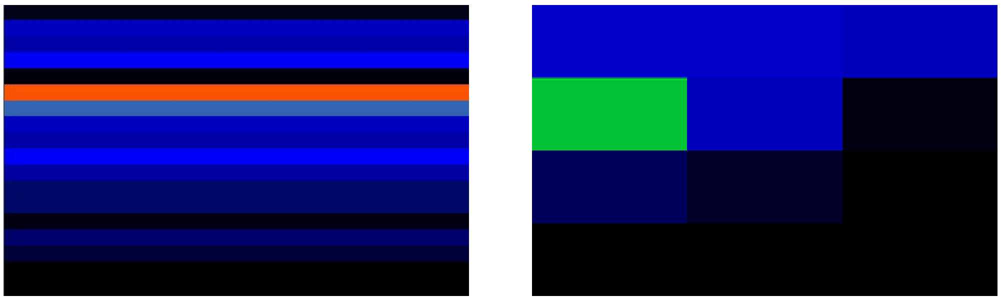

(b) A tiled image from the encoding algorithm of [1]

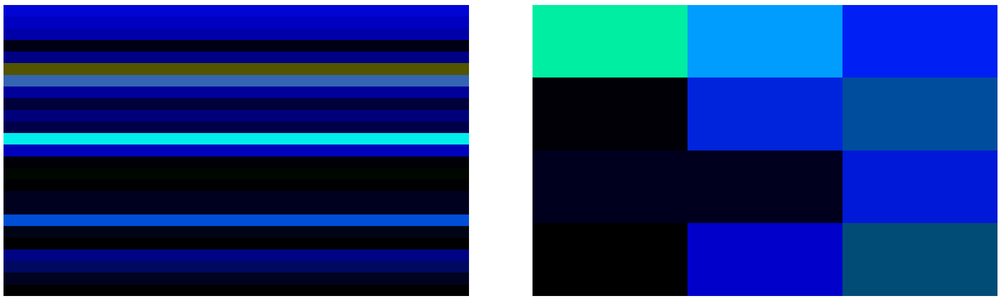
Fig. 5.3 A Visual Comparison of the Images for 5G Dataset

A visual representation of the images generated by the Net2I algorithm and the encoding algorithm used in [1], are presented in Fig.5.3 for 5G dataset, Fig.5.4 for UNSW NB-15 dataset whilst the images generated from the InSDN dataset are represented in Fig. 5.5. The NeT2I algorithm generates the images depicted in Fig.5.3a, Fig.5.4a, and Fig.5.5a which consists of one-dimensional horizontal lines with a variable x value and a fixed y value. Each line in the generated PNG, encompasses a traffic feature where as an IP address is distributed amongst four lines in the PNG. The images depicted in Fig.5.3b, Fig.5.4b, and Fig.5.5b are



(a) An image from the NeT2I algorithm (b) A tiled image from the encoding algorithm of [1]

Fig. 5.4 A Visual Comparison of the Images for UNSW NB-15 Dataset



(a) An image from the NeT2I algorithm (b) A tiled image from the encoding algorithm of [1]

Fig. 5.5 A Visual Comparison of the Images for InSDN Dataset

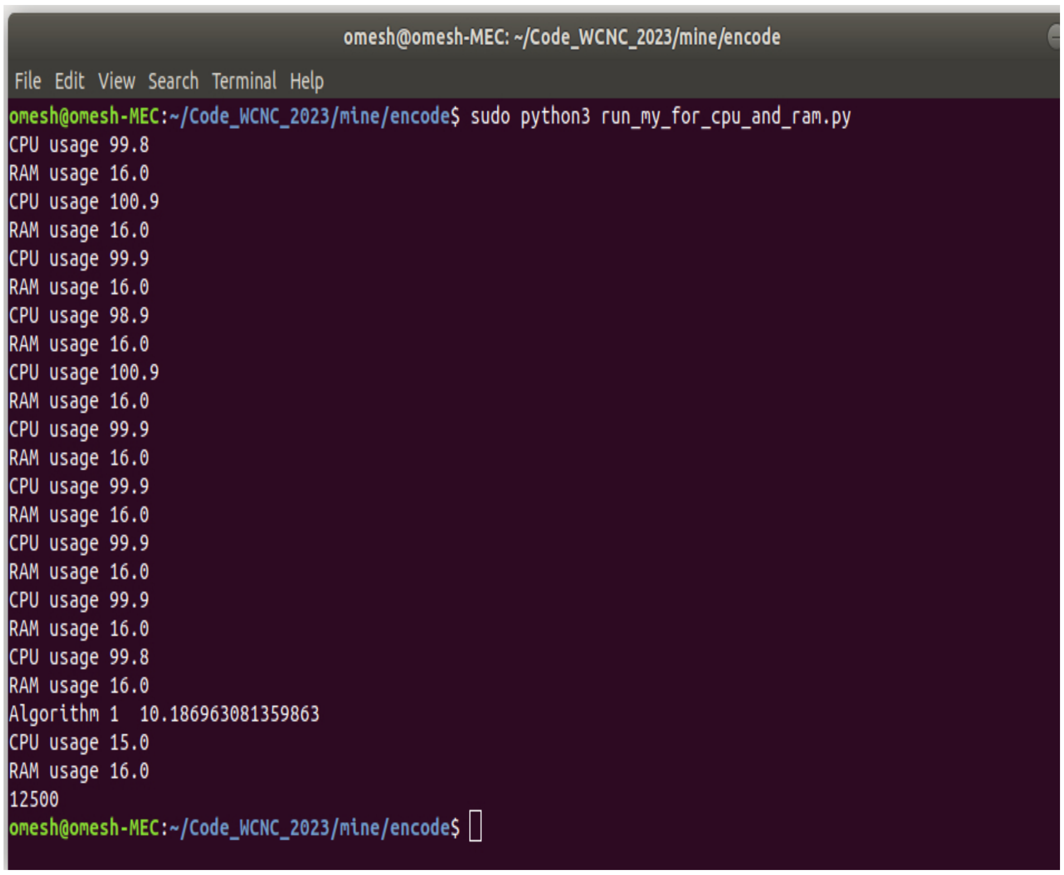
generated from the algorithm presented in [1]. These images are comprised of both x and y coordinates used to describe a tiled image for each feature present in the network traffic file.

5.4.2 Computational Complexity

Tables 5.5, 5.6, and 5.7 present the time of execution, CPU utilisation and memory utilisation of the NeT2I and I2NeT algorithms and the encoding and decoding algorithms used in [1]. A screenshot of the NeT2I execution has been depicted in Fig. 5.6.

Time for Execution

As the number of images remained uniform throughout, we formulated execution time per image, based on the total execution time, since our task of image creation from network traffic remains an Aperiodic Task [201]. From the collected results shown in Tables 5.5, 5.6, and 5.7 the NeT2I and I2NeT algorithms resulted in a smaller total execution time than the encoding and decoding algorithms of [1] for the same dataset which produced a reduction from 28s


omesh@omesh-MEC: ~/Code_WCNC_2023/mine/encode
File Edit View Search Terminal Help

```
omesh@omesh-MEC:~/Code_WCNC_2023/mine/encode$ sudo python3 run_my_for_cpu_and_ram.py
CPU usage 99.8
RAM usage 16.0
CPU usage 100.9
RAM usage 16.0
CPU usage 99.9
RAM usage 16.0
CPU usage 98.9
RAM usage 16.0
CPU usage 100.9
RAM usage 16.0
CPU usage 99.9
RAM usage 16.0
CPU usage 99.9
RAM usage 16.0
CPU usage 99.9
RAM usage 16.0
CPU usage 99.9
RAM usage 16.0
CPU usage 99.8
RAM usage 16.0
Algorithm 1  10.186963081359863
CPU usage 15.0
RAM usage 16.0
12500
omesh@omesh-MEC:~/Code_WCNC_2023/mine/encode$ 
```

Fig. 5.6 Computational Complexity (time, CPU, and RAM) during NeT2I

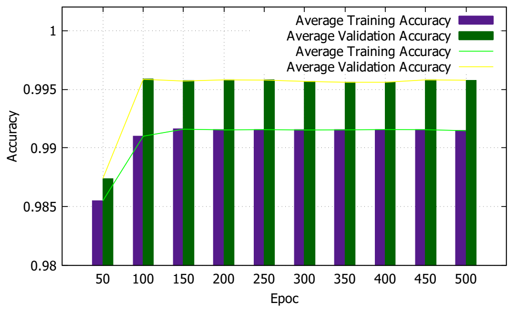
to 10s (5G Dataset), 27s to 6s (UNSW NB-15 Dataset), and 30s to 10s (InSDN Dataset) (encoding algorithm of [1] and NeT2I) for both malicious and non-malicious traffic. The decoding algorithm of [1] and I2NeT, utilised 28s and 6s (5G Dataset), 29s and 7s (UNSW NB-15 Dataset), and 30s and 6s (InSDN Dataset) for malicious traffic and for non-malicious traffic, respectively.

CPU Utilisation

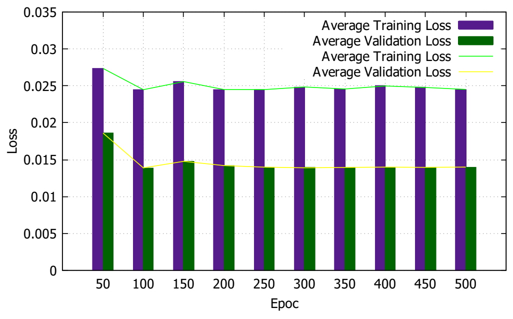
Given the global interpreter lock and single-threaded execution of Python code, we observe that the code, irrespective of the algorithm (NeT2I, I2NeT, encoding and decoding algorithm of [1]) used 100% of the CPU resources per image, however, the NeT2I and I2NeT algorithms utilised the CPU for a smaller window as seen in the total execution time in Tables 5.5, 5.6, and 5.7. The same observations were made across all three datasets.

Memory Utilisation

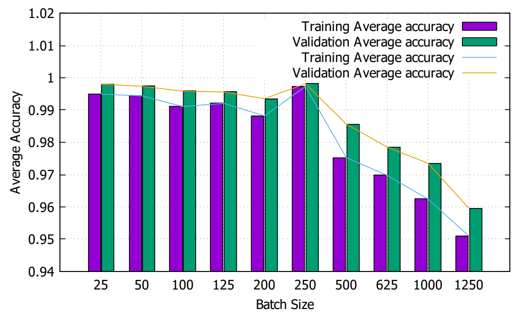
NeT2I utilised 20.2% of memory while the encoding algorithm of [1] utilised 27% memory. During the decode stage, I2NeT utilised 21% memory while the decoding algorithm of [1] utilised 28% of memory for the 5G dataset as described by Table 5.5. 19% and 28% of memory utilisation was recorded by NeT2I and encoding of [1], and 20% and 29% of memory for the I2NeT and decoding of [1], respectively, for the UNSW NB-15 dataset. InSDN dataset recorded 20.8% and 29.4% for NeT2I and encoding of [1], whilst the I2NeT and decoding of [1] recorded 20% and 29.8% of memory utilisation.



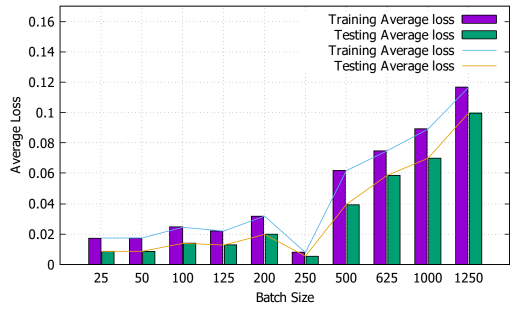
(a) Accuracy - Different epocs



(b) Loss - Different epocs



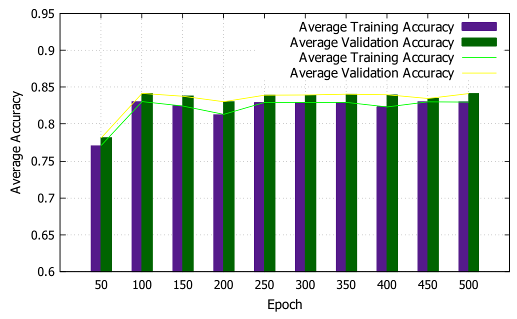
(c) Accuracy - Different batch sizes



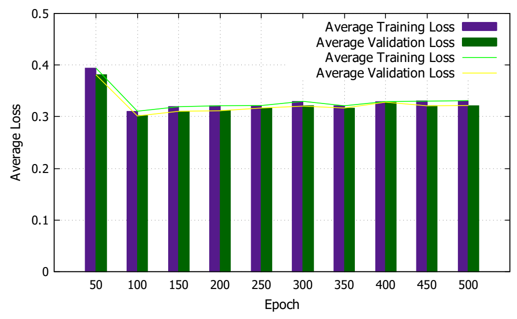
(d) Loss - Different batch sizes

Fig. 5.7 Training and Validation Data from NeT2I algorithm from the CNN for the 5G Dataset

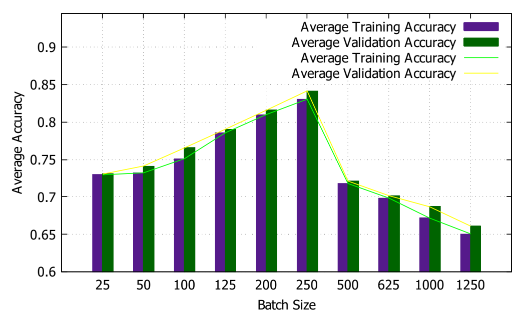
To establish the optimal batch size, divisors that can evenly divide the number of images in both the training and testing dataset were chosen, i.e: 50, 100, 125, 250, 500, 625 and 1250. In order to determine the most accurate epoch, values between 50, 100, 150, 200, 250, 300, 350, 400, 450 and 500 were chosen. The batch size and the epoch with the lowest amount of loss (a scalar value that compares the target and predicted values) and the highest accuracy rate were identified as optimal values. Upon completion of successful execution under the aforementioned batch sizes and epoch, for both datasets, CNN3L performed with



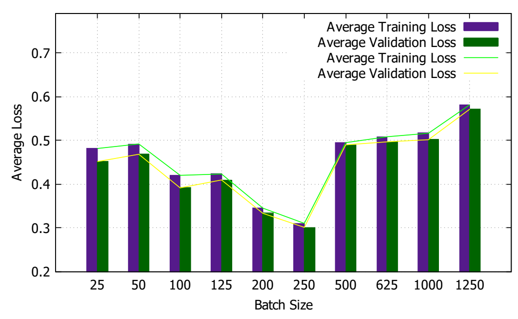
(a) Accuracy - Different epocs



(b) Loss - Different epocs



(c) Accuracy - Different batch sizes



(d) Loss - Different batch sizes

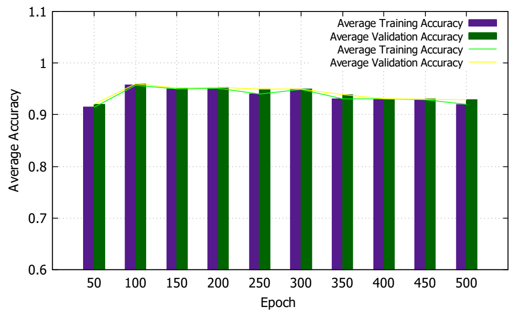
Fig. 5.8 Training and Validation Data from encoding algorithm [1] from the CNN for the 5G Dataset

significant results (high accuracy and low loss) when the batch size was 250 and the epoch was set to 100.

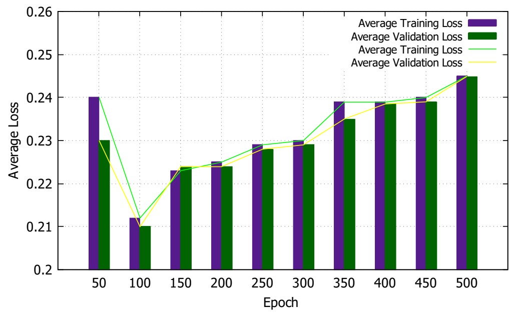
Optimal batch size and epoch for 5G Dataset

Fig. 5.7 present the average accuracy and loss collated for the various execution of CNN3L, to determine the optimal batch size and epoch, for the images generated from the 5G dataset, employing NeT2I. When the epoch value was incremented, the loss of training and information leakage [202], caused the average accuracy and the average loss to deviate from the optimal state. Figs. 5.7a and 5.7b, recorded that the CNN3L algorithm is at its optimal performance when the epoch value is at 100. Similarly, Figs. 5.7c and 5.7d, highlighted that the batch size 250 recorded the highest accuracy and the lowest loss for the CNN3L.

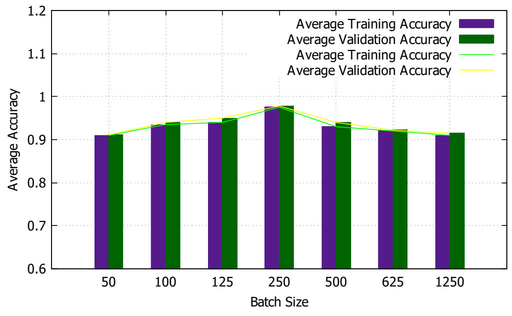
Fig. 5.8 present the results related to batch sizes and epoch when images were created from the encoding algorithm employed by [1]. Fig. 5.8a and 5.8b the highest accuracy and the lowest loss when the epoch value was at 100. Fig. 5.8c and Fig. 5.8d, present the data collated for different batch sizes of images sent to the algorithm for training and validation. The batch size 250 performed with the lowest loss and the highest accuracy.



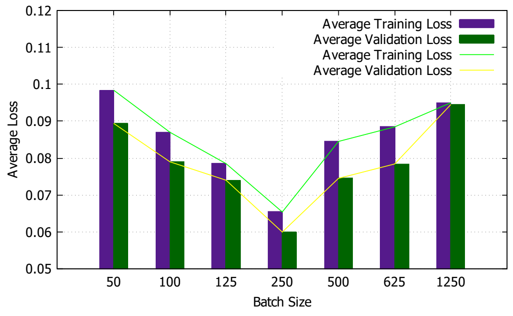
(a) Accuracy - Different epocs



(b) Loss - Different epocs



(c) Accuracy - Different batch sizes



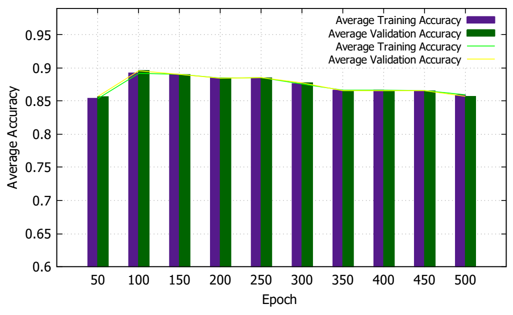
(d) Loss - Different batch sizes

Fig. 5.9 Training and Validation Data from NeT2I algorithm from the CNN for the UNSW NB-15 Dataset

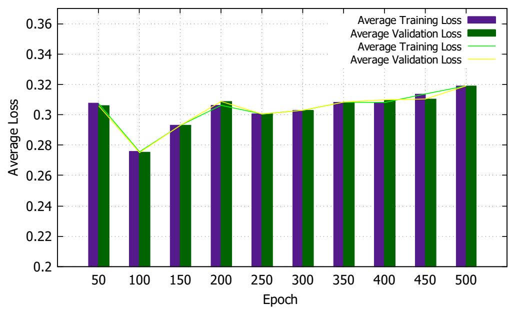
Optimal batch size and epoch for UNSW NB-15

Fig. 5.9 present the average accuracy and loss collated for the various executions of the CNN3L algorithm in order to determine the optimal batch size and the epoch, for the images created from the UNSW NB-15 dataset. In this iteration, the images were created using the NeT2I algorithm. By careful observation of Fig. 5.9a and Fig. 5.9b, epoch 100 performed the highest accuracy and the lowest loss. Fig. 5.9c and Fig. 5.9d, present the data collated for different batch sizes of images sent to the algorithm for training and validation. The batch size 250 performed with the lowest loss and the highest accuracy. As these two parameters (batch size and epoch) have to be decided from the initial experiment, we ran batch size experiments first with the epoch set as 100. Upon selecting the optimal batch size, we ran experiments pertaining to deciding the optimal epoch.

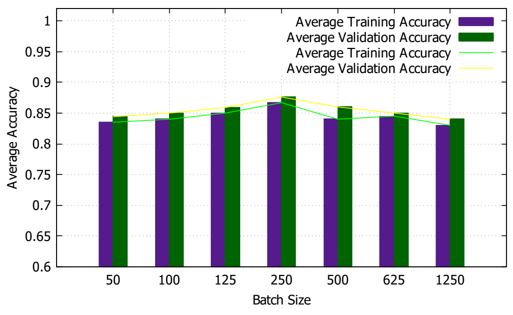
Fig. 5.10 present the results related to batch sizes and epoch when the images were created from the encoding algorithm employed by [1]. The number 100 was chosen for the epoch as the optimal value since that iteration of execution resulted in the lowest loss and highest accuracy. Fig. 5.10a and Fig. 5.10b present this result. Similarly, the optimal batch



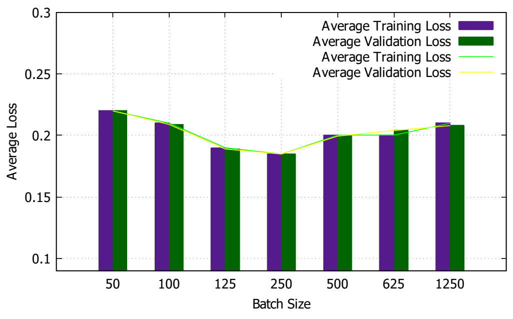
(a) Accuracy - Different epocs



(b) Loss - Different epocs



(c) Accuracy - Different batch sizes



(d) Loss - Different batch sizes

Fig. 5.10 Training and Validation Data from encoding algorithm of [1] from the CNN for the UNSW NB-15 Dataset

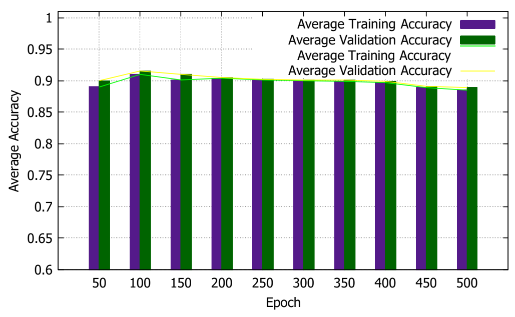
size was chosen as 250 for the images created from the UNSW NB-15 dataset when the encoding algorithm of [1] was employed. Fig. 5.10c and Fig. 5.10d present this result.

Optimal batch size and epoch for InSDN

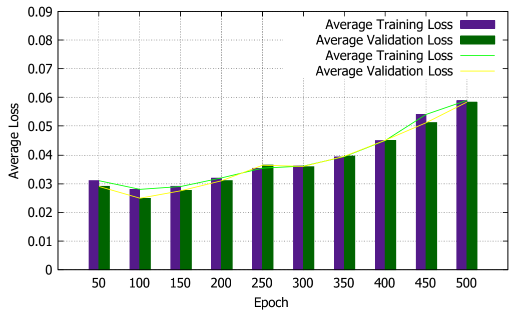
Similar to the UNSW NB-15 dataset, optimal values for batch sizes and epoch were also chosen for the images derived from the InSDN dataset. Images generated from both the NeT2I and encoding of [1] were used.

Fig. 5.11 presents the data collated for the experiments pertaining to both batch size and epoch for images generated from the NeT2I algorithm. Fig. 5.11a present the accuracy while Fig. 5.11b present the loss for different epoch. Accuracy and loss recorded their optimal values when the CNN3L algorithm was executed for 100 epoch. Fig. 5.11c and Fig. 5.11d present the optimal batch size as 250. The CNN3L produced the lowest loss and the highest accuracy when the batch size was set to 250 and epoch to 100.

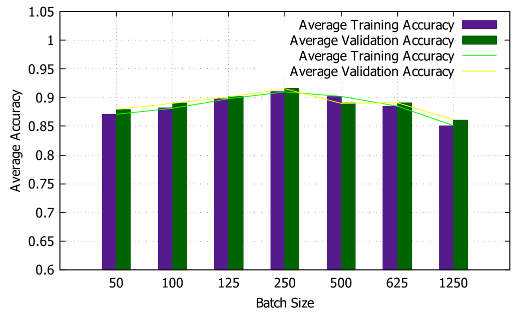
Fig. 5.12 present the data for the images generated from the encoding algorithm of [1] when used in CNN3L algorithm. Similar to previous instances, batch size and the epoch were evaluated by employing the previously mentioned divisors. Fig. 5.12a and Fig. 5.12b present the accuracy and loss for different epoch. The epoch 100 was chosen as the optimal value



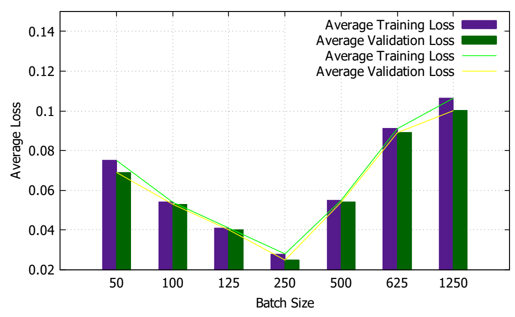
(a) Accuracy - Different epocs



(b) Loss - Different epocs



(c) Accuracy - Different batch sizes



(d) Loss - Different batch sizes

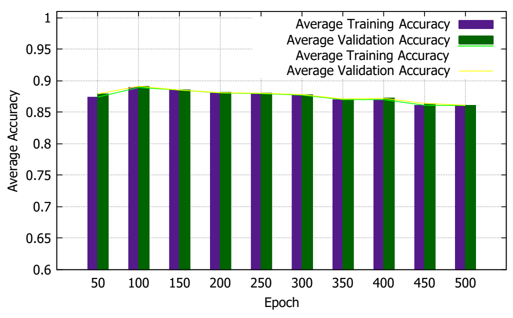
Fig. 5.11 Training and Validation Data from NeT2I algorithm from the CNN3L for the InSDN Dataset

since CNN3L registered the lowest loss and the highest accuracy. Fig. 5.12c, and Fig. 5.12d present that the batch size 250 was the optimal for the images generated from [1] encoding algorithm when applied to the CNN3L algorithm. The optimal values employed for training and testing CNN3L have been presented in Table 5.8.

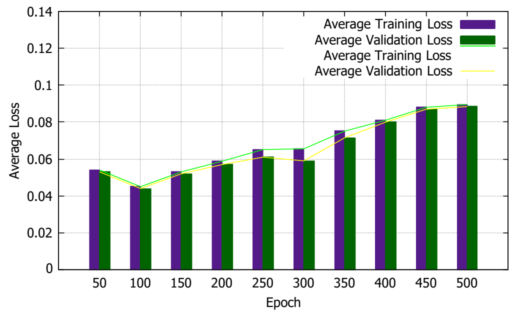
5.4.3 Training and Validation

Figs. 5.14a, 5.14b, 5.15a, 5.15b, 5.16a, and 5.16b present the accuracy and loss acquired from training and validation of the CNN3L algorithm with the optimal batch size and epoch for the 5G dataset, UNSW NB-15 dataset and InSDN dataset respectively. Lines depicted in blue and green, present the accuracy and loss for the CNN3L algorithm created using NeT2I. Similarly, the lines depicted in the red and black present the accuracy and loss created using the encoding algorithm from [1].

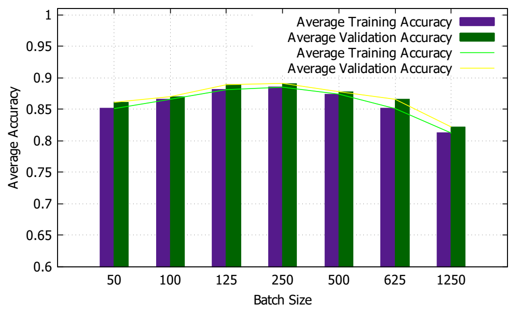
By using the configurations available in Table 5.8, a minimised loss was recorded with high accuracy. By observing lines of blue, green, red and black in Figs. 5.14a, 5.14b, 5.15a, 5.15b, 5.16a, and 5.16b, we can state that our model had minimal overfitting given the marginal difference between training and validation lines for both accuracy and loss. For the



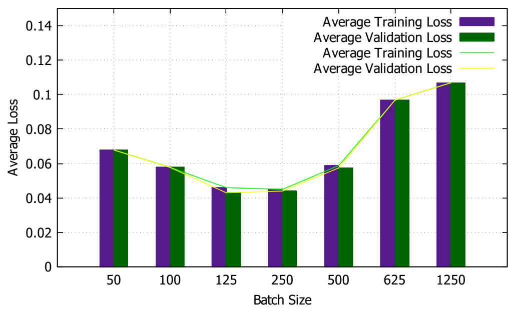
(a) Accuracy - Different epocs



(b) Loss - Different epocs



(c) Accuracy - Different batch sizes



(d) Loss - Different batch sizes

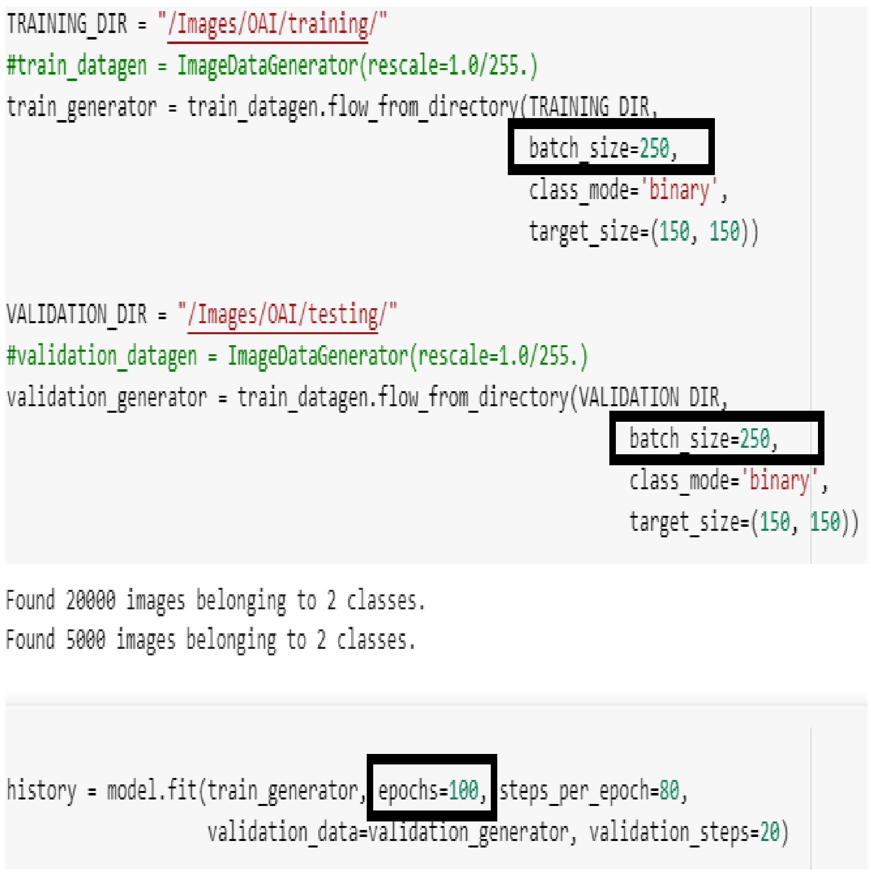
Fig. 5.12 Training and Validation Data from encoding algorithm of [1] from the CNN3L for the InSDN Dataset

5G dataset, observing epoch 80 in (Fig. 5.14a), which recorded an accuracy of 0.9669 and 0.9671, meant that the model didn't experience overfitting. We can also state that our model reached a stable performance state after epoch 80 for the UNSW NB-15 dataset, by observing the slope for accuracy (Fig. 5.15a) and loss (Fig. 5.15b) during training and validation. Upon reaching a stable performance state, our model continued to show a minimum difference between training and validation (at epoch 80 for NeT2I + CNN3L training and validation values were 0.964050026 and 0.965800018), suggesting that our model did not experience much overfitting.

Corresponding configurations in Table 5.8 were used once more to the CNN3L algorithm to the images derived from the InSDN dataset. Recorded minimal loss and high accuracy can be observed in Fig. 5.16a and Fig. 5.16b. Another observation that we can make is that similar to the previous model, this iteration too experienced minimal overfitting. This can be noticed by the marginal difference between training and validation lines for both accuracy and loss. Furthermore, our model during the underpinned experiment experienced a stable state after epoch 60, by observing the slope for accuracy (Fig. 5.16a) and loss (Fig. 5.16b) during training and validation. Upon reaching a stable performance state, our model continued to show a minimum difference between training and validation (at epoch 60

Variable	Parameter
Convolutional2D	3 Layers
MaxPooling	3 Layers
Activation	Sigmoid
Optimiser Function	RMSprop
Batch Sizes	250
Epoch	100
Kernel Size	3*3
Loss Function	Binary Cross Entropy
Output classes	2

Table 5.8 Specification of the CNN3L Algorithm



```
TRAINING_DIR = "/Images/OAI/training/"
#train_datagen = ImageDataGenerator(rescale=1.0/255.)
train_generator = train_datagen.flow_from_directory(TRAINING_DIR,
                                                    batch_size=250,
                                                    class_mode='binary',
                                                    target_size=(150, 150))

VALIDATION_DIR = "/Images/OAI/testing/"
#validation_datagen = ImageDataGenerator(rescale=1.0/255.)
validation_generator = train_datagen.flow_from_directory(VALIDATION_DIR,
                                                         batch_size=250,
                                                         class_mode='binary',
                                                         target_size=(150, 150))

Found 20000 images belonging to 2 classes.
Found 5000 images belonging to 2 classes.

history = model.fit(train_generator, epochs=100, steps_per_epoch=80,
                    validation_data=validation_generator, validation_steps=20)
```

Fig. 5.13 Training and Testing the CNN3L using optimal specifications as per Table 5.8.

for NeT2I + CNN3L training and validation values were 0.995850027 and 0.996200025) whilst the difference between training and validation loss were 0.01306867,0.010818329, suggesting that our model did not experience much overfitting.

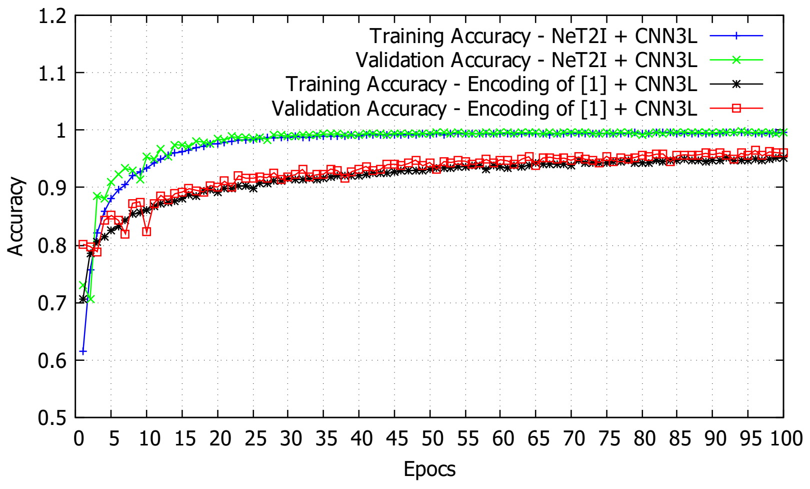
The accuracy of the CNN3L model has been presented in Fig. 5.15a for both NeT2I + CNN3L and the encoding algorithm of [1] + CNN3L where the images were generated from the UNSW NB-15 dataset. The former (NeT2I + CNN3L) outperformed the latter with an average of 95% and 97% whereas the latter (encoding algorithm of [1] + CNN3L) produced an average of 86% and 88% for training and validation accuracy. This supports our claim that using NeT2I to create the images leads to greater accuracy with the CNN3L model during training and validation. As loss is the summation of errors in the model during training and validation, the model trained and tested using the NeT2I images resulted in a lower loss compared to the encoding algorithm of [1] as seen in Fig. 5.15b created using the NeT2I resulted in a lower averaged loss of 6% and 5% during training and testing whereas the image dataset from the encoding algorithm of [1] resulted an averaged loss of 18% and 17% during training and validation, thus highlighting that the NeT2I + CNN3L outperformed the encoding algorithm of [1] + CNN3L in both accuracy and loss.

The accuracy and loss in both training and validation for the CNN3L algorithm from the images derived from the InSDN dataset have been represented in Fig. 5.16a and Fig. 5.16b for both NeT2I + CNN3L and the encoding algorithm of [1] + CNN3L. The former (NeT2I + CNN3L) outperformed the latter with an average of 96% and 97% whereas the latter (encoding algorithm of [1] + CNN3L) produced an average of 90% and 91% for training and validation accuracy. This supports our claim that using NeT2I to create the images leads to greater accuracy with the CNN3L model during training and validation. As seen in Fig. 5.16b created using the NeT2I resulted in a lower averaged loss of 4% and 6% during training and testing whereas the image dataset from the encoding algorithm of [1] resulted in an averaged loss of 19 % and 18 % during training and validation, thus highlighting that the NeT2I + CNN3L outperformed the encoding algorithm of [1] + CNN3L in both accuracy and loss across both UNSW NB-15 and InSDN datasets.

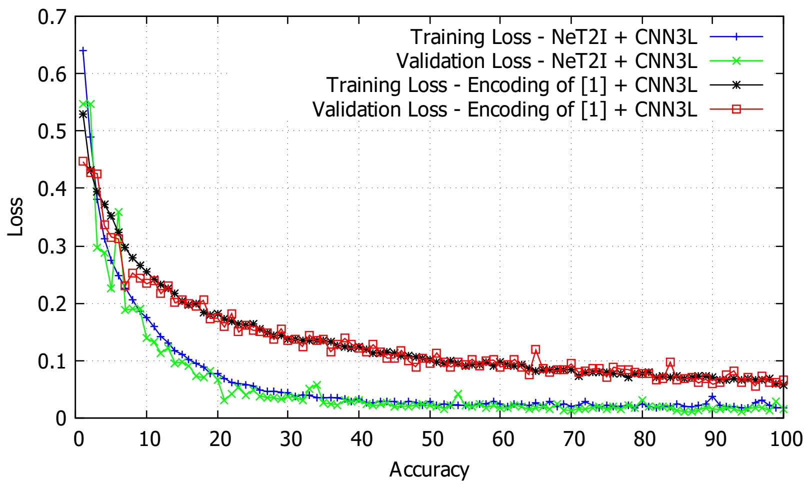
5.4.4 Evaluation of Detection

Algorithm	Accuracy	Prediction	Recall	F1-Score
NeT2I + CNN3L	0.97	0.96	0.97	0.96
Encoding of [1] + CNN3L	0.91	0.9	0.92	0.91

Table 5.9 Confusion Matrix of the Detection Algorithm for the 5G dataset.

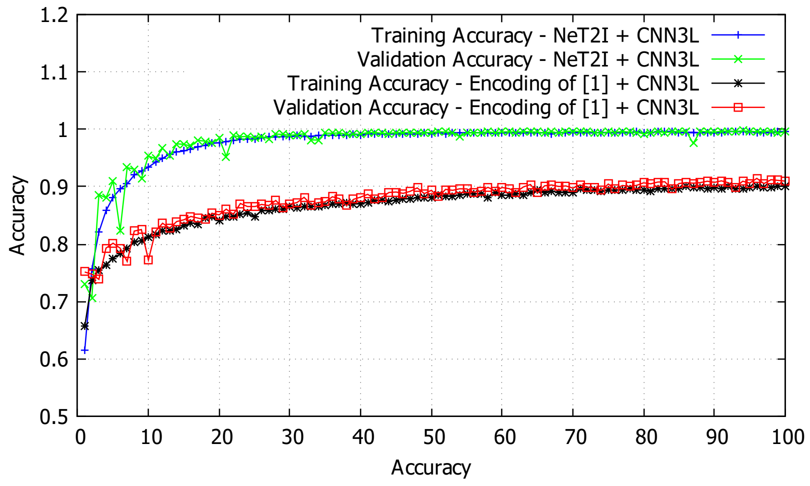


(a) Training and Validation Accuracy

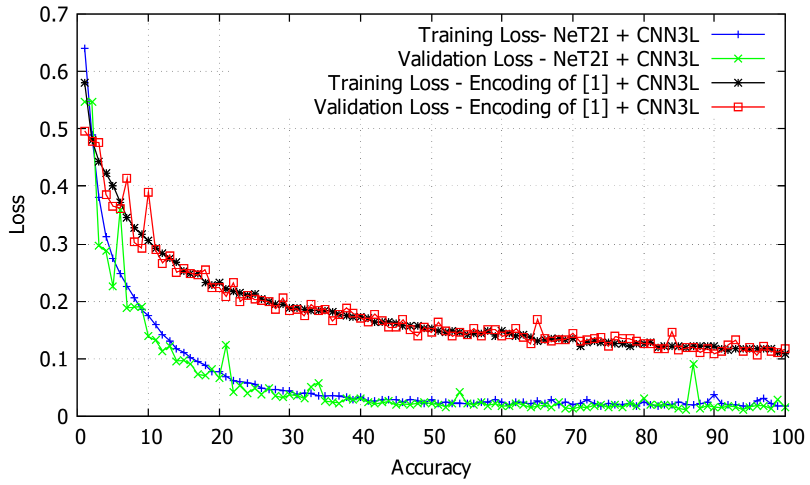


(b) Training and Validation Loss

Fig. 5.14 Training and Validation Data from the Proposed 5G dataset using NeT2I and encoding of [1]

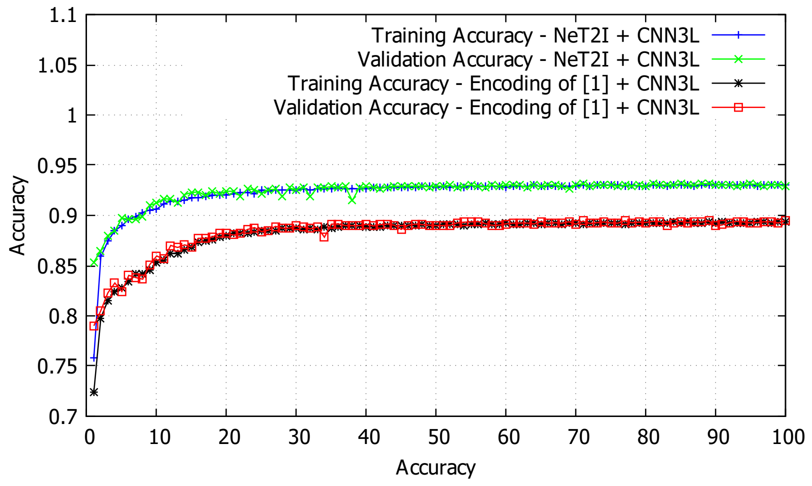


(a) Training and Validation Accuracy

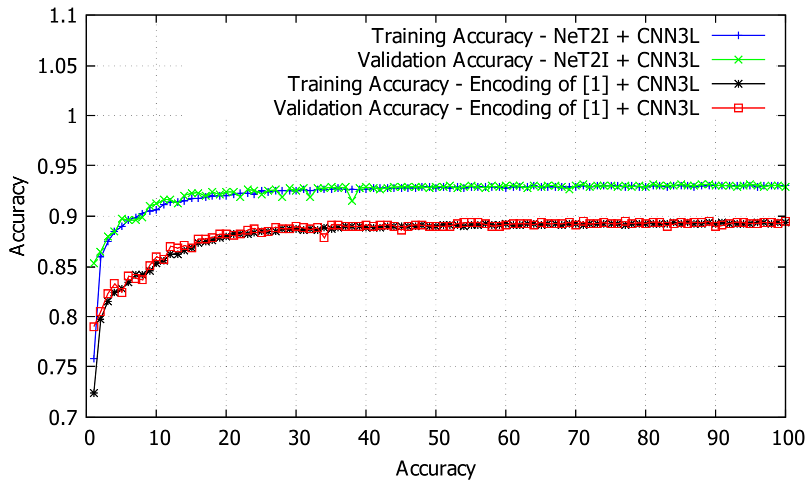


(b) Training and Validation Loss

Fig. 5.15 Training and Validation Data from the UNSW NB-15 dataset



(a) Training and Validation Accuracy



(b) Training and Validation Loss

Fig. 5.16 Training and Validation Data from the InSDN dataset

Algorithm	Accuracy	Prediction	Recall	F1-Score
NeT2I + CNN3L	0.96	0.95	0.97	0.95
Encoding of [1] + CNN3L	0.89	0.9	0.89	0.89

Table 5.10 Confusion Matrix of the Detection Algorithm for the UNSW NB-15 Dataset.

Algorithm	Accuracy	Prediction	Recall	F1-Score
NeT2I + CNN3L	0.96	0.95	0.97	0.95
Encoding of [1] + CNN3L	0.93	0.92	0.91	0.9

Table 5.11 Confusion Matrix of the Detection Algorithm for the InSDN Dataset.

The results of our experiments are shown in Tables 5.9, 5.10, and 5.11. The employment of the NeT2I algorithm and the CNN3L algorithm, achieved a higher accuracy in comparison to the encoding mechanism employed by the authors in [1] and the CNN3L algorithm. The applied methodologies of detecting malicious traffic for the 5G dataset with a detection rate of 97% for the NeT2I and CNN3L with 91% for the encoding of [1] and CNN3L for the same features. UNSW dataset with the same 11 features as [1] has resulted in a higher detection rate of 96% from NeT2I algorithm + CNN3L and 89% from encoding from [1] + CNN3L respectively. Similarly, the InSDN dataset recorded an accuracy of 96% and 93% for NeT2I + CNN3L, as opposed to the encoding of [1] + CNN3L. We compared these results with those produced by other methodologies, for example, 47.19% in [203], binary classification in ML (linear regression (74.3%), Naive Bayes (77.3%), k Nearest Neighbour (81%) and radial basis function in support vector machine (65.3%)) and deep neural network (80.1%) in [204] 37.15% in [205], and 81.42% [206].

NeT2I + CNN3L achieved 95%, 97% and 95% respectively for prediction, Recall and F1-Score while encoding from [1] + CNN3L achieved 90%, 89% and 89% respectively. As achieving a 100% accuracy would signify an overfitting of data in neural networks, the achieved 96% can be considered as having achieved cross-validation in our methodology due to augmentation and increase of varied training data [207].

5.5 Chapter Summary

In this chapter, we developed and presented a new method to encode network traffic into RGB images with less computation against a well-established method of encoding. We conducted this research to advance the security of the 5G-MEC infrastructure by employing an effective

IDS based on CNN with low resource utilisation. Next chapter we aim to implement the NeT2I, encoding of [1], and CNN3L algorithms in the 5G-MEC mobile telecommunication testbed, that we discussed in Chapter 4, where aim to answer our final research question: How do we apply DL in real-time to intrusion detection in 5G-MEC mobile telecommunication testbed?

Chapter 6

Real-Time Application of Deep Learning Intrusion Detection in 5G-MEC

6.1 Overview

This chapter presents a new Network Intrusion Detection System (NIDS) employing the algorithms (NeT2I, CNN3L, and I2NeT) presented in Chapter 5. The new NIDS has been launched in a 5G-Multi-Access Edge Computing (MEC) Mobile Telecommunication Testbed (Chapter 4) in real-time. We evaluate the proposed NIDS in terms of its computational complexity in for example: time, memory and CPU utilisation for the signature-detection, encoding and decoding algorithms, and its accuracy and loss during training and detection.

6.2 Introduction

Multi-Access Edge Computing (MEC) and its respective methods of deployment have been presented in [2, 208] and as awareness and interest in 5G-MEC has grown within academia and industry, research based on resource allocation, energy awareness and network slicing has also grown [209]. Despite the popularity of research in the field of NIDS technology, many applications remain based on signature-based methodologies for the detection of malicious traffic [4, 89–91]. Due to the increased volume of network data, lack of in-depth monitoring and granularity, as well as the diverse range of data types and protocols [4, 93] in a 5G mobile network, traditional signature-based NIDS systems may be less effective [94, 95].

As demonstrated in [120, 121], a signature-based NIDS can perform better when combined with an intelligent agent (Machine Learning or Deep Learning). However, a bottleneck for the application of ML to NIDS problems is created by the inability to handle

Experiments
RQ4: Real-time Implementation of Intrusion Detection in the 5G Mobile Telecommunication Testbed
1. Study of the existing hypothetical two-staged IDSs
2. Study of network traffic collection techniques
3. Implement a signature detection algorithm
4. Incorporate encoding algorithm after signature detection
5. Evaluate the computational complexity of the encoding algorithm against literature
6. Link Google Colab
7. Apply the Confusion Matrix

Table 6.1 Experiments undertaken in this Chapter

data with higher dimensions, the associated time overhead in training, the need for large amounts of limited dimensional data, the labour-intensive process of identifying relevant data, and the heterogeneity of 5G data [4, 95]. Authors at [89, 124] reviewed the latest advancements in NIDS using ML/DL approaches in which the authors state that the DL techniques have received the most attention from both industry and academia. Hence we apply a DL approach towards the detection of malicious traffic in real-time.

Our research is focused towards securing the MEC architecture at the edge of the network, hence algorithms will be bound by the computational complexity (RAM, CPU and time complexity) since the MEC architecture will inherently possess low processing power. Based on the literature, we proposed a multi-stage Real-Time Deep Leaning Network Intrusion Detection System (RTDL-NIDS), developed by conducting the following experiments, mentioned in Table 6.1.

6.3 Real-Time Deep Leaning Network Intrusion Detection System (RTDL-NIDS)

The proposed NIDS: Real-Time Deep Learning Network Intrusion Detection System (RTDL-NIDS) is illustrated in Fig. 6.1, launched in the 5G-MEC mobile telecommunication testbed [152], as depicted in Fig. 6.4. The RTDL-NIDS has been implemented as a soft real-time application for intrusion detection. This is due to its execution on general-purpose hardware and the limited resource availability [210] that is expected to have on MEC nodes. The deadline of execution is also set by the user to accommodate different traffic volumes, which addendum to the soft real-time execution. The testbed encompasses components of the Radio

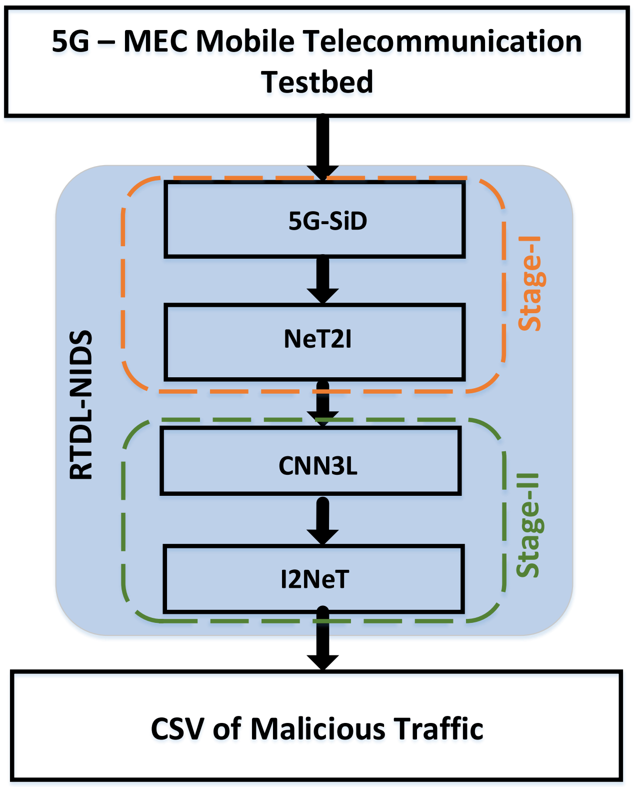


Fig. 6.1 Real-Time Deep Leaning Network Intrusion Detection System (RTDL-NIDS)

Access Network (RAN), Core Network (CN), MEC Network, and User Equipment (UE). The NIDS is launched in the MEC-IDS node, where real-time, low-computational signature detection is carried out, followed by the employment of algorithms capable of data encoding and detection using a DL agent that we developed in Chapter 5.

6.3.1 Stage-I

The RTDL-NIDS has two stages. Stage-I employs three algorithms. They are 5G-Signature Detection algorithm (5G-SiD), NeT2I from Chapter: 5.2.1, and Fuse_Google_Drive.

Stage-I: 5G-Signature Detection algorithm (5G-SiD)

The 5G-SiD algorithm functions in the following manner.

- MEC-IDS node stores 5G-SiD and listens to the traffic for the MEC-UDP and captures using PyShark [211] and this is the input to 5G-SiD.
- Captured pcapng file is processed into a CSV file.

- Resulting CSV file from the previous step, undergoes several Py functions and conducts a signature-detection and labelling of network traffic occurs.
- Produces an output CSV, containing suspicious network traffic.

Stage I: NeT2I

NeT2I from Chapter: 5.2.1 functions in the following manner:

- NeT2I reads the CSV file generated from 5G-SiD.
- NeT2I Converts integer and string values to RGB format
- RGB values are used to generate images
- Save PNG images

Stage I: Algorithm : Fuse_Google_Drive

Fuse_Google_Drive functions in the following manner:

- Link Google Drive and local file system
- Upload the PNG images from NeT2I to Google Drive for the Google Collaboratory to access.

This concludes Stage I.

6.3.2 Stage II

Stage II includes two algorithms. They are CNN3L and I2NeT from Chapter: 5.2.2.

Stage II: CNN3L

CNN3L from Chapter: 5.2.3 functions in the following manner:

- CNN3L reads a designated folder in Google Drive
- Uploads the images generated via NeT2I
- Classifies traffic between malicious and non_malicious.

Stage II: I2NeT

I2NeT from Chapter: 5.2.2 functions in the following manner:

- Reads the images
- Translates images to RGB format
- Maps RGB values to network features
- Generate a CSV file

Upon successful generation of the output CSV from I2NeT, and from the images which were classified as malicious from the CNN3L, we can gather information about malicious sources. In the next section, we will present a detailed explanation of the developed 5G-SiD algorithm, which helps us conduct quick signature detection.

6.4 5G-MEC Signature Detection (5G-SiD Algorithm)

The pseudo-code in Algorithm 4:5G-MEC Signature Detection Algorithm (5G-SiD), describes the steps involved in the new signature detection algorithm. We chose a signature detection algorithm to be implemented at the initial stage of detection, considering the following rationale.

1. **Resource Efficiency**: Signature detection mechanisms are computationally effective as they compare the system against a collection of known threats. This mechanism of signature detection has been corroborated by authors where devices have constraints on resources [7] and energy [118]. Implementation of a Signature-based detection system for intrusion detection alleviates the need to provide dedicated edge nodes with higher computational power.

2. **Low False Positive rate:** The research presented by authors in [109, 110] highlights the effectiveness of the application of signature detection towards intrusion detection problem, towards the identification of known threats. With the effective management of signatures, the system can be optimised to detect malicious intent.

3. **Quick Response:** Research presented by [111, 112] highlight the quick response time for detection or prevention generated by the intrusion detection systems that are categorised as signature detectors. The ease of matching signatures against a database, and the lack of observing baseline statistics have paved the way towards this outcome.

4. **Ease of Management:** Research by authors in [111, 112, 116] highlight the ease of including new signatures to detect into the respective signature-based intrusion detection system. Through effective management of the signatures to match, the IDS can efficiently detect malicious intent.

The 5G-SiD algorithm was developed considering the above nationalities. The computational complexity of the algorithm was monitored during development and execution. This was carried out to ensure that the algorithm maintain a low computational requirement. The 5G-SiD provided a quick response time with a higher accuracy when malicious traffic was present in the network. Also, as per [7, 66], a quick detection in the initial stage of intrusion detection is crucial to maintain stringent performance requirements for future networks. The py function, responsible for matching new signatures, was developed with ease of management in mind, particularly for including new signatures to match.

The 5G-SiD conducts signature detection in the following manner. Dedicated Py functions calculate the variance of source/destination port variations and packet size to detect malicious attacks such as DoS, DDoS, and port scan. Similarly, another Py function inspects ws.col.Info for certain flags (ACK, RST-ACK) that have been set in the network traffic, to detect SYN flood attacks. The detection in Stage-I, alleviates the computational workload for the CNN-based algorithm (CNN3L), leaving the detection time for malicious network traffic, reduced. Feature extrapolation has been described in Table 6.2.

Protocol	**Features Extrapolated**						
ICMP	frame.number	ip.src	eth.src	ip.dst	ip.proto	frame.len	ws.col.Info
TCP	frame.number	ip.src	eth.src	ip.dst	ip.proto	tcp.dstport tcp.srcport	tcp.window size value frame.len ws.col.Info
UDP	frame.number	ip.src	eth.src	ip.dst	ip.proto	udp.dstport udp.srcport	udp.length ws.col.Info
SSH	frame.number	ip.src	eth.src	ip.dst	ip.proto	tcp.dstport tcp.srcport	ssh.packet.length frame.len ws.col.Info
SCTP	frame.number	ip.src	eth.src	ip.dst	ip.proto	sctp.srcport sctp.dstport	sctp.port frame.len ws.col.Info
ARP	frame.number	arp.src.proto.ipv4	eth.src	arp.dst.proto.ipv4	arp.isprobe	frame.len	ws.col.Info
GTP (TCP/ UDP/ ICMP)	frame.number	ip.src	eth.src	ip.dst	ip.proto	dstport srcport	tcp.window size value frame.len length ws.col.Info

Table 6.2 Collected Features and Flags from the Network Traffic

The functions in the 5G-SiD algorithm are described in detail below.

1. **Initiate PyShark:** Upon initialisation of the RTDL-NIDS, PyShark listens and captures network traffic to a pcapng file for a defined period of time. In Algorithm 4: line 3,

Algorithm 4: 5G-Signature Detection Algorithm (5G-SiD)

```
Input: Network Traffic
while SIGINT == FALSE do
    Function Initiate PyShark():
        Collect for n seconds
        SSH .pcapng to MEC:IDS
    Function Segregate Traffic():
        Filter Traffic using TShark
        Generate a CSV
    Function Label Traffic():
        while ReadCSV do
            Function Check Packet Size():
                Result ≡ Equation 6.1
                /* See Equation 6.1                                  */
                if Result ≡1 then
                    Write Label_1≡ Int(1)
                else
                    Write Label_1≡ Int(0)
            Function cal_Src_Port_Variation ():
                for 3 lines in Source_Column do
                    Variation ≡ Equation 6.2
                    /* See Equation 6.2                              */
                    if Variation ≡ 1 then
                        Write Label_2≡ Int(1)
                    else
                        Write Label_2≡ Int(0)
            Function cal_Dest_Port_Variation():
                for 3 lines in Dest_Column do
                    Variation ≡ Equation 6.3
                    /* See Equation 6.3                              */
                    if Variation ≡ 1 then
                        Write Label_3≡ Int(1)
                    else
                        Write Label_3≡ Int(0)
            Function Check_Flags():
                Result ≡ Equation 6.4
                /* See Equation 6.4                                  */
                if Result ≡ 1 then
                    Write Label_4≡ Int(1)
                else
                    Write Label_4≡ Int(0)
            Function Additional Rules():
                /* Insert Additional Rules to detect Signatures      */
            Function Write Label_5():
                Result ≡ Equation 6.5
                /* See Equation 6.5                                  */
                if Result ≡ 1 then
                    Write Label_5≡ Int(1)
                else
                    Write Label_5≡ Int(0)
        Return CSV
    System Shutdown
```

collects network traffic for n number of seconds, which the user can define to facilitate a soft real-time execution. We ran our algorithm continuously to collect network traffic in 10s intervals. In the testbed illustrated in Fig.6.4, the algorithm listens to the interface on the BMv2 switch connecting CN and RAN, to bring the detection as close as possible to the source.

2. **Traffic Segregation and reorder:** Upon collection of a large pcapng file, network traffic was segregated based on the protocol and was ordered accordingly. The primary reason behind this was to apply a function at a later stage that can detect variations of source and destination port numbers. Upon filtering the traffic, another Py function converts the collected pcapng file into a CSV file with information such as flow ID, source IP, source MAC, destination IP, source port, destination port, protocol number, packet length and information column. Based on the above fields, traffic labelling is conducted by successor functions.

3. **Traffic Labelling:** For detection which is faster with quick and efficient processing of network packets, we employed a signature-based IDS system, due to the resource limitation in MEC nodes. Following is the description of the nested functions.

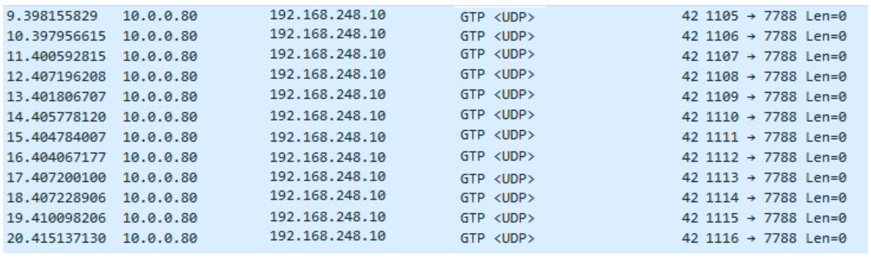
4. **Check Packet Size:** In the predecessor function, we omitted the traffic that was generated via the USRP device towards the RAN. Given this exclusion, packets exceeding the standard size may be due to a network intrusion. Also, network packets with 0 size will also be due to malicious intent. Hence, we employ a function that can create a label for this type of traffic upon reading the CSV file. We used the integer 1 for label_1 for traffic with packet size exceeding 1500 or packets with size registered as 0.

$$label\_1 = \begin{cases} 1 & \text{if } (Packet\_size > 1500) \,||\, (Packet\_size = 0) \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

5. **Calculate Source Port Variations:**

$$label\_2 = \begin{cases} 1 & \text{if } \left(\sum_{i=1}^{3n} \frac{(SP_{(i)} + SP_{(i+1)} + SP_{(i+2)})}{3(SP_i)} \right) > 1 \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

For this function, we process the number of packets to the maximum multiple of 3. The remaining 1 or 2 packets are left unprocessed and written to the output file directly. This function is feasible since the assignment of port numbers is always in uniform



```
9.398155829   10.0.0.80   192.168.248.10   GTP <UDP>   42 1105 → 7788 Len=0
10.397956615  10.0.0.80   192.168.248.10   GTP <UDP>   42 1106 → 7788 Len=0
11.400592815  10.0.0.80   192.168.248.10   GTP <UDP>   42 1107 → 7788 Len=0
12.407196208  10.0.0.80   192.168.248.10   GTP <UDP>   42 1108 → 7788 Len=0
13.401806707  10.0.0.80   192.168.248.10   GTP <UDP>   42 1109 → 7788 Len=0
14.405778120  10.0.0.80   192.168.248.10   GTP <UDP>   42 1110 → 7788 Len=0
15.404784007  10.0.0.80   192.168.248.10   GTP <UDP>   42 1111 → 7788 Len=0
16.404067177  10.0.0.80   192.168.248.10   GTP <UDP>   42 1112 → 7788 Len=0
17.407200100  10.0.0.80   192.168.248.10   GTP <UDP>   42 1113 → 7788 Len=0
18.407228906  10.0.0.80   192.168.248.10   GTP <UDP>   42 1114 → 7788 Len=0
19.410098206  10.0.0.80   192.168.248.10   GTP <UDP>   42 1115 → 7788 Len=0
20.415137130  10.0.0.80   192.168.248.10   GTP <UDP>   42 1116 → 7788 Len=0
```

Fig. 6.2 Spoofed DDoS

increments of 1 or 2 [212]. Since a malicious user can create multiple connections to the same destination as a means of a DoS or DDoS attack with or without spoofed IP addresses, employing this function in Algorithm 4: 5G-Signature Detection Algorithm: line 16 which encompasses the equation available in 6.2, will aid in identifying an attack. Fig.6.2 depicts the Wireshark capture when a Spoofed DDoS scan attack is being conducted by an adversary, to a UDP server at port 7788. The following example which is derived from the Fig.6.2, assigns 1 to *label_2*.

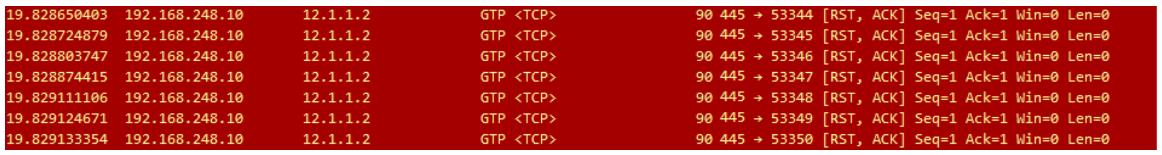
$$label\_2 = \left(\sum_{i=1}^{3n} \frac{(1105+1106+1107)}{3*1105} \right)$$
$$label\_2 = 1.0009$$
$$label\_2 = 1.0009 > 1$$
$$\therefore label\_2 = 1$$

In equation 6.2, *i* denotes the number of lines and *SP* denotes the source port number. Integer 1 is assigned to label_2 if the calculated value is greater than 1, indicating that the source port variations occurred, highlighting an attack is underway.

6. **Calculate Destination Port Variations:**

$$label\_3 = \begin{cases} 1 & \text{if } \left(\sum \frac{(DP_{(i)}+DP_{(i+1)}+DP_{(i+2)})}{3(DP_i)} \right) > 1 \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

When a malicious user is initiating a port scan attack, the source IP and destination IP will remain static whilst the destination port will increase continuously. Fig.6.3 depicts a Wireshark capture when a port scan attack is underway. To detect a port scan attack, we employed the equation available in 6.3 in Algorithm 4: 5G-Signature Detection: line 23. Similar to the predecessor function if the calculated value is greater than 1, we assigned integer 1 to the label_3. The example below which was derived from Fig.6.3, *label_3* is assigned with 1 to mark malicious activity of destination port variation.



```
19.828650403  192.168.248.10        12.1.1.2            GTP <TCP>           90 445 → 53344 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19.828724879  192.168.248.10        12.1.1.2            GTP <TCP>           90 445 → 53345 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19.828803747  192.168.248.10        12.1.1.2            GTP <TCP>           90 445 → 53346 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19.828874415  192.168.248.10        12.1.1.2            GTP <TCP>           90 445 → 53347 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19.829111106  192.168.248.10        12.1.1.2            GTP <TCP>           90 445 → 53348 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19.829124671  192.168.248.10        12.1.1.2            GTP <TCP>           90 445 → 53349 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19.829133354  192.168.248.10        12.1.1.2            GTP <TCP>           90 445 → 53350 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

Fig. 6.3 Port scan attack

$$label\_3 = \left(\sum_{i=1}^{3n} \frac{(53344+53345+53346)}{3*53344} \right)$$
$$label\_3 = 1.0000187$$
$$label\_3 = 1.0000187 > 1$$
$$\therefore label\_3 = 1$$

7. **Check Flags:**

   As the final stage of signature detection, we employed a function capable of reading the flags and information column on the traffic capture. The collated labels are mentioned in Table 6.2 along with their respective traffic type. The function in Algorithm 4: 5G-Signature Detection: line 30, which encompasses the equation 6.4 will set label_4 to integer 1 if one or more of the criteria are met: i.e [RST, ACK], Ack=1 or Seq=1.

$$label\_4 = \begin{cases} 1 & \text{if } (([\text{RST, ACK} = 1]) || (\text{Ack} = 1) || (\text{Seq} = 1)) \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

   Lastly, if any of the labels (label_1, label_2, label_3, or label_4) are set to integer 1, a final label was created in the function Algorithm 4: 5G-Signature Detection: line 37 to write integer 1 to label_5 using an OR operation as per the equation 6.5.

$$label\_5 = \begin{cases} 1 & \text{if } ((label\_1 = 1) \,||\, (label\_2 = 1) \,||\, (label\_3 = 1) \,||\, (label\_4 = 1)) \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

Completion of the above functions will create a CSV file, which carries traffic that is both malicious or susceptible to being detected as malicious. This CSV file serves as an input to the NeT2I algorithm, which translates network traffic into PNG images with RGB colours. The generated images are uploaded to the CNN3L function by the employment of the Fuse_Google_Drive algorithm, which utilised the ocaml_fuse library [213] to link the Google Drive to the local file system.

In the next section, we evaluate the RTDL-NIDS which encompasses the 5G-SiD, NeT2I, CNN3L, and I2NeT.

Algorithm 5: Fuse_Google_Drive

```
  Input: PNG
  Output: Detection Results
1 Function Connect to Google Drive():
2     Google Drive ocamlfuse
3     while SIGINT == FALSE do
4         while ocamlfuse==True do
5             image counter = os.listdir()
6             for each image in (image counter) do
7                 Upload _Images_To_CNN3L
8         System Shutdown
9     return Detection Results
```

6.5 Evaluation of the Proposed NIDS

6.5.1 5G-MEC Mobile Telecommunication Testbed

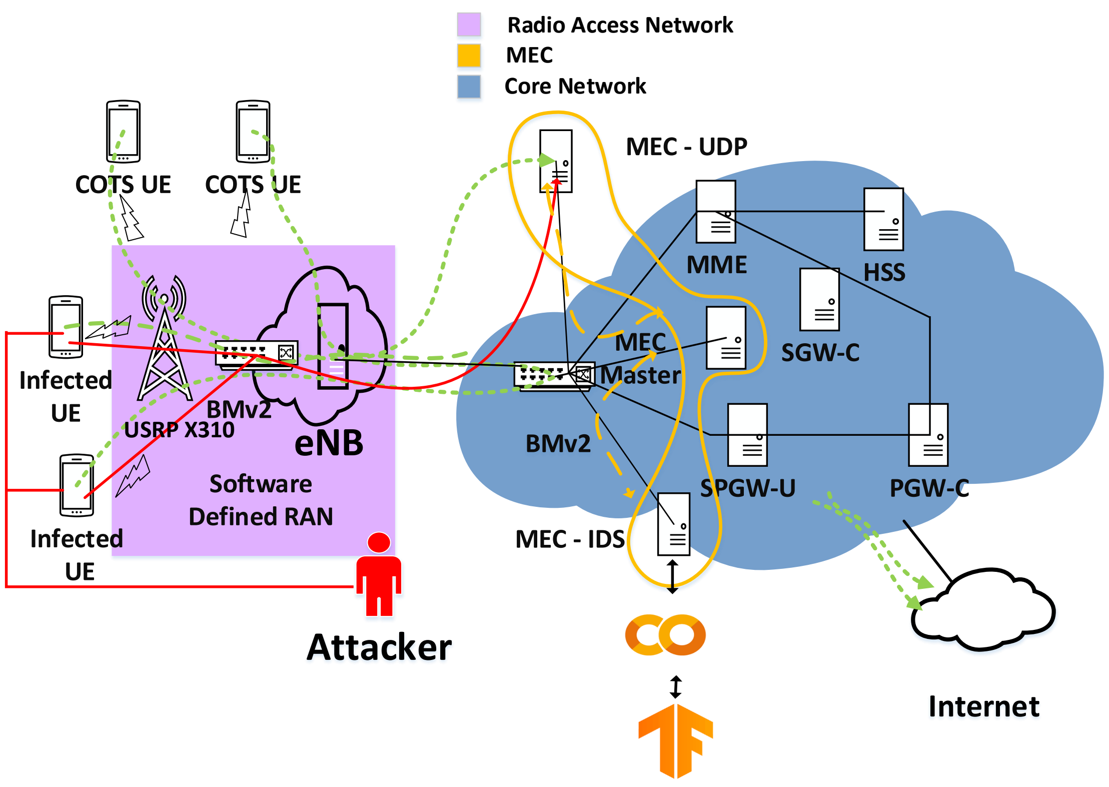


Fig. 6.4 Real-Time 5G Testbed with MEC-IDS and CNN

UE Number	Assignment	Source IP address	Spoofed IP address
UE 1	Non_Malicious	12.1.1.1	NA
UE 2	Non_Malicious	12.1.1.2	NA
UE 3	Non_Malicious	12.1.1.3	NA
UE 4	Non_Malicious	12.1.1.4	NA
UE 5	Non_Malicious	12.1.1.5	NA
UE 6	Non_Malicious	12.1.1.6	NA
UE 7	Malicious	12.1.1.7	10.0.0.80
UE 8	Malicious	12.1.1.8	10.0.0.81
UE 9	Malicious	12.1.1.9	192.168.20.23
UE 10	Malicious	12.1.1.10	172.168.10.32
UE 11	Malicious	12.1.1.11	172.168.10.33
UE 12	Malicious	12.1.1.12	172.168.10.34

Table 6.3 Assignment of UEs and IP addresses

The testbed in Chapter 4.2 has been expanded to accommodate RTDL-NIDS using CNN. See Fig.6.4. The testbed consists of three main elements: Core network (CN), Radio Access Network (RAN) and multi-access edge computing (MEC) deployment. RAN is denoted by the purple area in Fig.6.4, CN is denoted by the blue cloud, and the MEC deployment is depicted in yellow. The MEC architecture was instantiated using three UVT-Cloud environments, the MEC-Master, the MEC-UDP and MEC-IDS. All three nodes have been virtualized as UVT-Cloud environments running Ubuntu Bionic servers. Each node has been instantiated with 2GB of RAM, 10GB of HDD and 1 CPU core. This is to reflect the lightweight, low processing power, that MEC nodes possess. An iPerf UDP server has been initiated on the MEC Client node with the bind option, for the UE's to transfer UDP data.

MEC-IDS node listens to the network traffic of MEC-UDP. MEC-IDS node execute the RTDL-NIDS as per the Fig.6.1. MEC-IDS node has been connected to the Google Colaboratory environment. A prerequisite to the MEC-IDS is that the CNN3L has been trained and tested, and the predict function is awaiting images for detection. Once the images have been generated via the NeT2I, Google_Fuse_Drive algorithm uploads the images to the Google Drive, for the predict function to access. This mechanism provides the realisation of MEC, where services are launched in the Cloud as closest to the user. Since the CNN3L is not launched locally, the algorithm can be used to detect malicious traffic in any location of the MEC network.

Table 6.3 refers to the assignment of UEs and the IP addressing used for the purpose of this experiment. Given the limitation of dongles, we virtualised each connected laptop to

Original Network Traffic	RTDL-NIDS	NIDS of [1]
Flow ID	✓	✓
Source IP	✓	Cyphered
Source MAC	✓	×
Destination IP	✓	Cyphered
Source port	✓	✓
Destination port	✓	✓
Protocol number	✓	✓
Packet length	✓	✓
Acknowledgement	✓	✓
	Source Port Change	Source Port Change
	Destination Port Change	Destination Port Change
	Check Flags	Check Flags
	Label	Label

Table 6.4 Features extrapolated for RTDL-NIDS and NIDS of [1]

host two Virtual Machines, totalling the number of connected devices to three in each laptop. Hence we were able to connect 12 UEs to the testbed.

6.5.2 Existing Algorithms for Comparison

To evaluate the performance of the algorithms 5G-SiD, NeT2I, I2NeT, and CNN3L, we would like to compare our IDS with an existing IDS. The IDS presented in [1] selected is a recently published study selected, because of its use of 2D RGB images for network traffic, with the application of CNN for the detection of malicious traffic and for its high accuracy. The workflow of the proposed NIDS (RTDL-NIDS) and the NIDS of [1] can be found in Fig.6.5.

1. Upon completion of the 5G-SiD, a cypher algorithm as per [1] was employed to encrypt the source IP address and destination IP address. MAC address was dropped from the list of features. This was done in order to be aligned with any other work, that may employ a publicly available dataset since information pertaining to MAC addresses is omitted. The extrapolated features and how they were applied in the RTDL-NIDS against the NIDS of [1] can be found in Table 6.4.

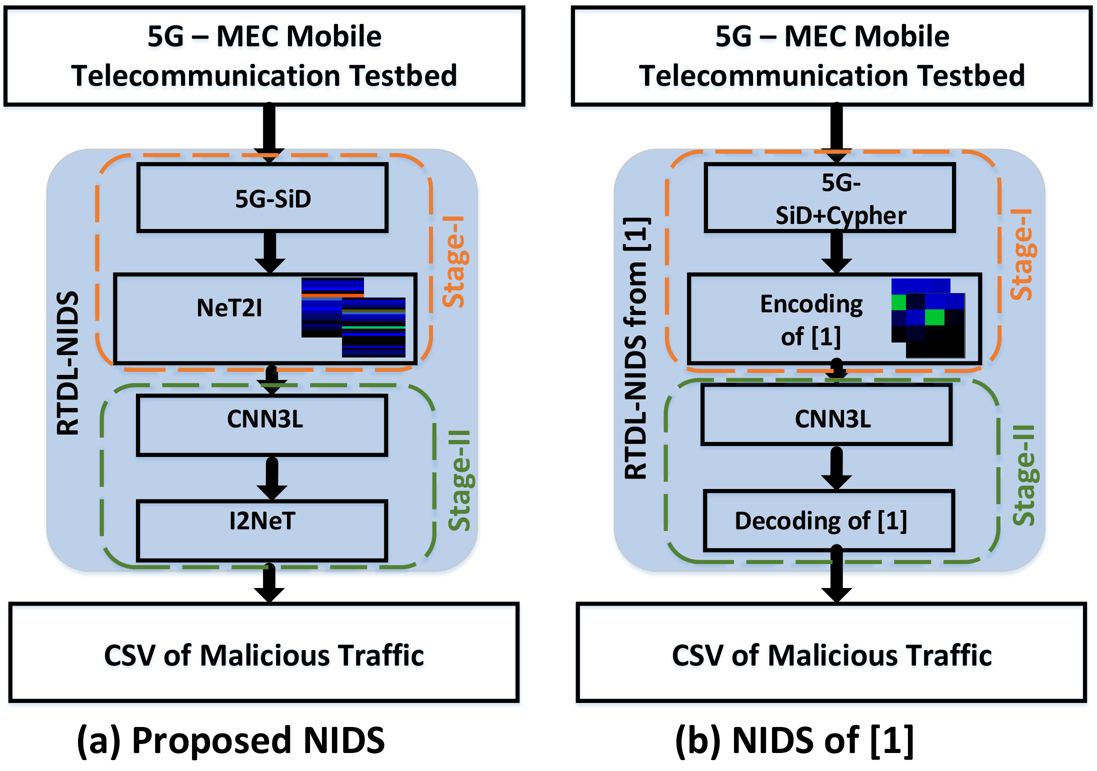


Fig. 6.5 Workflow and algorithms for RTDL-NIDS and NIDS of [1]

2. NeT2I is compared with the encoding algorithm of [1]. Both transfer traffic data to images for the detection of malicious traffic. The NeT2I algorithm produces one-dimensional RGB images while the encoding of [1] generates tiled RGB images.

3. The CNN3L algorithm is used for both IDSs. Although a two-layer CNN is used in [1], due to the recorded performance gain in employing three convolutional layers as opposed to two layers [140], we employed, for comparative purposes CNN3L for both IDSs.

4. The I2NeT algorithm was evaluated against the decoding algorithm of [1], although not explicitly given in their work. This algorithm is capable of generating a CSV file for the associated network traffic from a collection of tiled PNG images by reading the pixel value of each tile in the image and mapping the value to its respective integer value. Since the encoding algorithm of [1] masked the IP address of the source and destination, a simple RGB to Integer function was employed to decode all tiles in the image into a corresponding a CSV file.

6.5.3 Evaluation Metrics

Matrices for Detection

We evaluated our 5G-SiD algorithm based on the confusion matrix. As the computational complexity is evaluated, the confusion matrix will provide a statistic with regard to its applicability in detecting malicious attacks. Next, we evaluated the CNN3L algorithm based on configurations, for example, convolutional layers, epochs and batch sizes, in order to understand the percentage deviation that occurs in the accuracy of the detection algorithm when different configurations are employed.

We considered evaluating our CNN3L algorithm in terms of overfitting. This occurs when the CNN model performs with significant results for the training data and can not perform well for the validation data. We evaluated overfitting by observing and calculating the difference between training and validation lines for both accuracy and loss.

The following metrics [5] were used in terms of accuracy (A), precision (P), recall (R) and F1-Score (F1) to evaluate the proposed CNN detection algorithm.

$$A = \frac{TP+TN}{TP+FP+TN+FN} \tag{6.6}$$

$$P = \frac{TP}{TP+FP} \tag{6.7}$$

$$R = \frac{TP}{TP+FN} \tag{6.8}$$

$$F1 = \frac{P*R*2}{P+R} \tag{6.9}$$

TP stands for the number of positively predicted attacks, FP stands for negatively predicted attacks on non-malicious traffic, TN represents non-malicious traffic that was correctly predicted as normal, and finally, FN stands for malicious traffic that was predicted as normal.

Computational Complexity

For evaluating the RTDL-NIDS and the NIDS of [1], we compared the computational complexity of each in terms of the time of execution, CPU and RAM. Since the MEC nodes possess low resources, an algorithm that conducts a signature-based detection and converts network traffic into images, with minimal resource utilisation will be advantageous.

- Time for execution

  To accurately measure the execution time, we used the Python Time library. As mentioned in reference [199], determining the exact execution time of a program is crucial, as it is better than using a rough estimate. Variations in execution time can impact performance and efficiency, which is particularly significant in low-resource environments, such as in intrusion detection.

- CPU Utilisation

  To measure CPU utilization, we used the Python psutil library to collect the percentage of usage for the algorithm. Our code uses the standard single-threaded Python execution with global interpreter lock, which means that the algorithms can only be executed on one single CPU thread at a time. Given that our algorithms are expected to be executed on an MEC node with limited processing capabilities, it is essential that they effectively use the CPU resources for an optimal period of time.

- Memory Utilisation

  As mentioned in reference [200], memory allocation is not as precise as CPU allocation or usage. To determine memory allocation for our algorithms, we used the Python library memory_profiler. Computer systems often over-allocate memory to a process for efficient execution, and since garbage collection doesn't happen immediately, an average value for the memory allocation is used for comparison of performance.

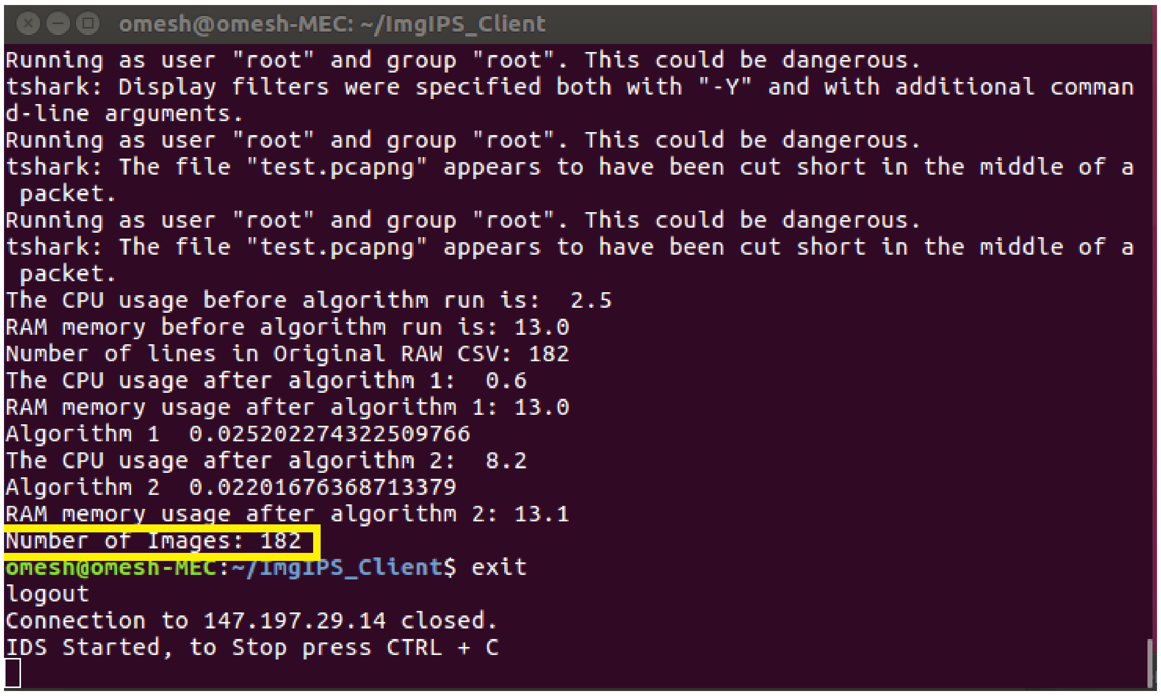
6.6 Results and Analysis

We first present results pertaining to the testing of the RTDL-NIDS, and the encoded images generated from the real-time network traffic in the 5G-MEC mobile telecommunication testbed from the NeT2I and the encoding of [1]. Next, we present the computational complexity accumulated from the 5G-SiD and 5G-SiD+Cypher. Lastly, we compare the new RTDL-NIDS against the NIDS of [1], in terms of computational complexity and evaluation of detection. We conclude this section with the results related to detection.

6.6.1 Testing the workflow of RTDL-NIDS

RTDL-NIDS Execution

Fig.6.6 presents screenshots while the RTDL-NIDS launched in the 5G-MEC mobile telecommunication testbed which shows the continued execution of RTDL-NIDS. In the given iteration, the 5G-SiD has classified 182 lines of network traffic as suspicious.



```
Running as user "root" and group "root". This could be dangerous.
tshark: Display filters were specified both with "-Y" and with additional comman
d-line arguments.
Running as user "root" and group "root". This could be dangerous.
tshark: The file "test.pcapng" appears to have been cut short in the middle of a
 packet.
Running as user "root" and group "root". This could be dangerous.
tshark: The file "test.pcapng" appears to have been cut short in the middle of a
 packet.
The CPU usage before algorithm run is:  2.5
RAM memory before algorithm run is: 13.0
Number of lines in Original RAW CSV: 182
The CPU usage after algorithm 1:  0.6
RAM memory usage after algorithm 1: 13.0
Algorithm 1  0.025202274322509766
The CPU usage after algorithm 2:  8.2
Algorithm 2  0.02201676368713379
RAM memory usage after algorithm 2: 13.1
Number of Images: 182
omesh@omesh-MEC:~/ImgIPS_Client$ exit
logout
Connection to 147.197.29.14 closed.
IDS Started, to Stop press CTRL + C
```

Fig. 6.6 RTDL-NIDS launched in MEC-IDS node

NeT2I Output

The same number of images (i.e., 182 images) has been created as shown in Fig.6.7, by the execution of NeT2I. Fuse_Google_Drive has uploaded the images for the detection algorithm launched in the Google Colab environment.

CNN3L Output

Fig.6.8 depicts the detection of CNN3L occurring in real-time. Fuse_Google_Drive uploads the images generated from the NeT2I to the Google Drive for the CNN3L to access. Depicted screenshot (Fig.6.8), shows that the images are being detected in real-time by the algorithm.

6.6.2 Comparison of Encoded Images

A visual representation of the images generated by the Net2I algorithm and the encoding algorithm of [1], in the 5G-MEC mobile telecommunication testbed are presented in Fig.6.9. The NeT2I algorithm generated the images depicted in Fig.6.9a, which consist of one-dimensional horizontal lines with a variable x value and a fixed y value. Each line in the generated PNG, encompasses a traffic feature where as an IP address is distributed amongst

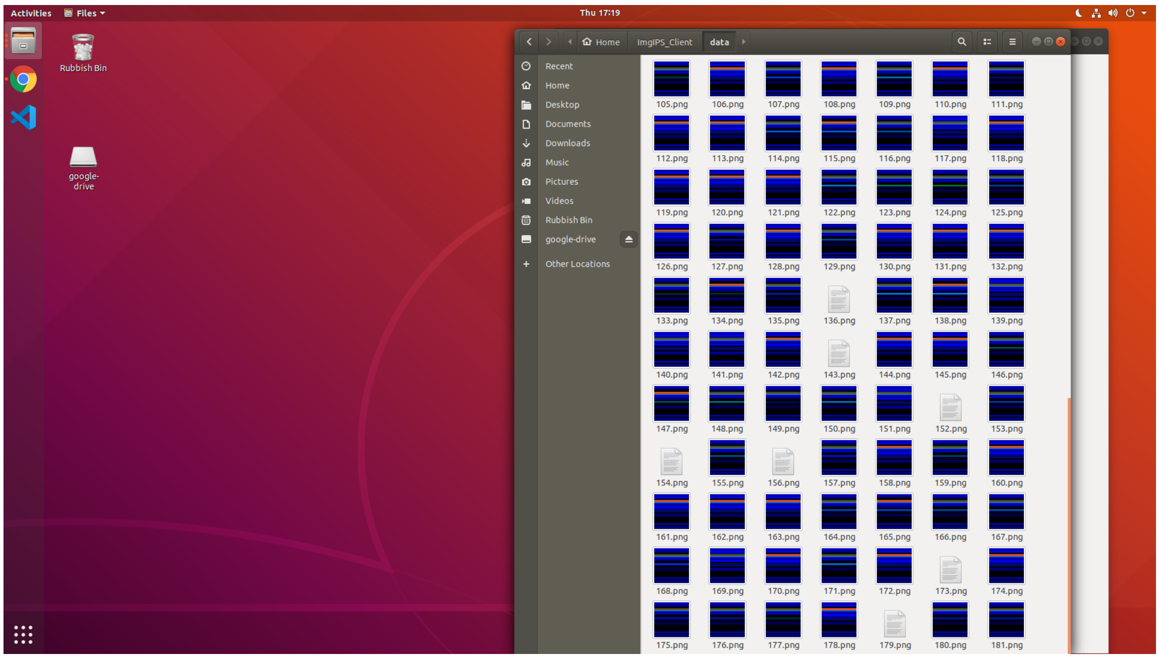


Fig. 6.7 Execution of RTDL-NIDS launched in MEC-IDS node

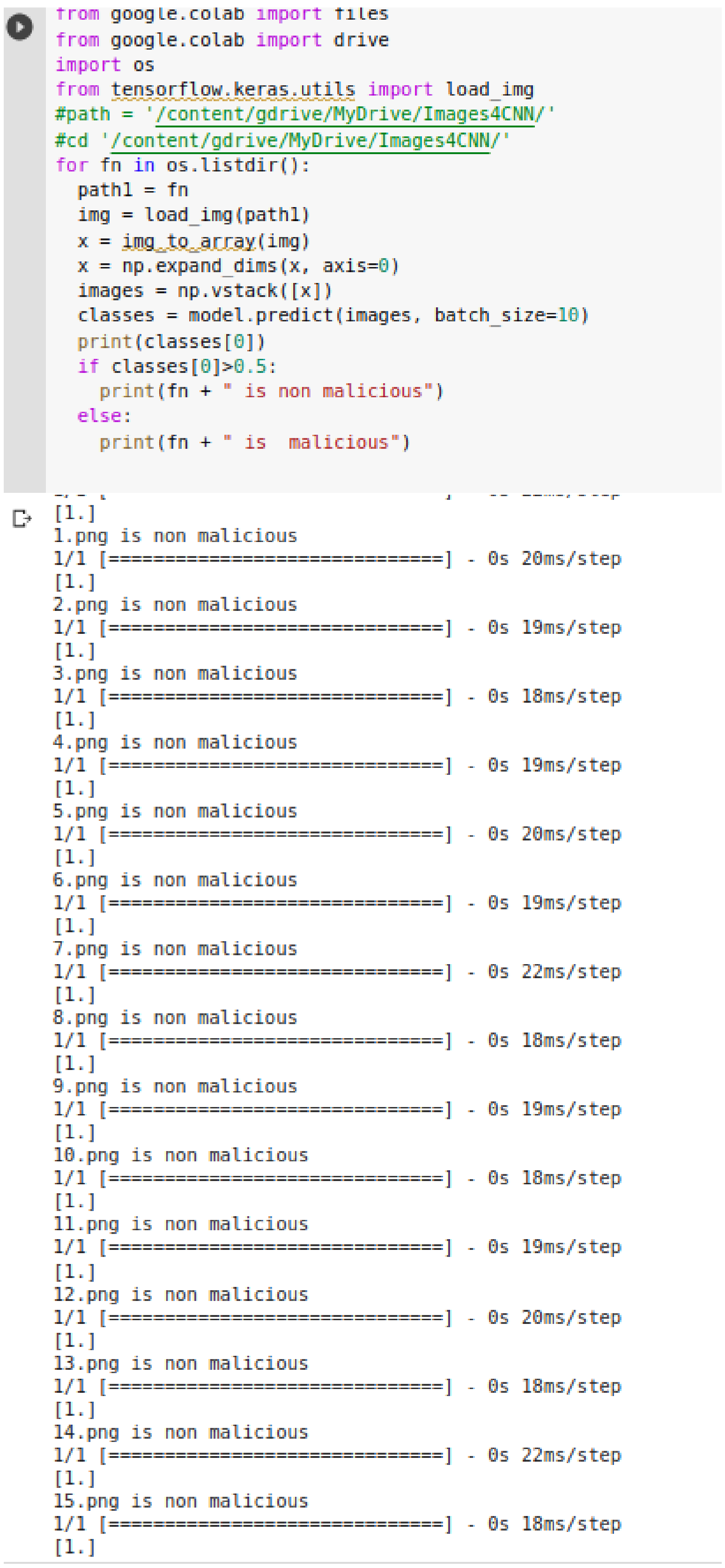
four lines in the PNG. The images generated from the encoding algorithm of [1], which consists of both x and y coordinates, have been represented in Fig. 6.9b.

6.6.3 Evaluation of Detection

The results of our experiments pertaining to the new signature detection algorithms are shown in Tables 6.5, and 6.6. We first present the confusion matrix for the newly developed signature detector, followed by the confusion matrix generated from the CNN3L algorithm when applied in real-time to the 5G-MEC Mobile Telecommunication Testbed.

Signature Based Detection (5G-SiD) in Stage-I

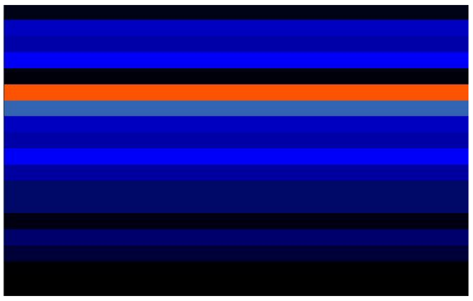
The results from the signature-based detection with the application of 5G-SiD are shown in Table 6.5. Table 6.5 presents the accuracy, precision, recall and F1 score for the malicious attacks that we conducted in the 5G-MEC mobile telecommunication testbed. Our algorithm performed with an accuracy of 85%, 82%, and 74% for port scan, TCP Syn Flood and DDoS. Prediction, Recall and F1-score also showed 87%, 98% and 92% for the port scan attack, whilst 85%, 96% and 91% for TCP SYN Floods, and 78%, 83% and 81% for the DDoS attack. Thus suggesting that the port scan attack was more accurately detected by our signature-based detection algorithm, than DDoS, or TCP SYN Floods. Given the nature of signature detection, it is not feasible to detect malicious traffic with an absolute state of



```
from google.colab import files
from google.colab import drive
import os
from tensorflow.keras.utils import load_img
#path = '/content/gdrive/MyDrive/Images4CNN/'
#cd '/content/gdrive/MyDrive/Images4CNN/'
for fn in os.listdir():
  path1 = fn
  img = load_img(path1)
  x = img_to_array(img)
  x = np.expand_dims(x, axis=0)
  images = np.vstack([x])
  classes = model.predict(images, batch_size=10)
  print(classes[0])
  if classes[0]>0.5:
    print(fn + " is non malicious")
  else:
    print(fn + " is  malicious")
```

```
[1.]
1.png is non malicious
1/1 [==============================] - 0s 20ms/step
[1.]
2.png is non malicious
1/1 [==============================] - 0s 19ms/step
[1.]
3.png is non malicious
1/1 [==============================] - 0s 18ms/step
[1.]
4.png is non malicious
1/1 [==============================] - 0s 19ms/step
[1.]
5.png is non malicious
1/1 [==============================] - 0s 20ms/step
[1.]
6.png is non malicious
1/1 [==============================] - 0s 19ms/step
[1.]
7.png is non malicious
1/1 [==============================] - 0s 22ms/step
[1.]
8.png is non malicious
1/1 [==============================] - 0s 18ms/step
[1.]
9.png is non malicious
1/1 [==============================] - 0s 19ms/step
[1.]
10.png is non malicious
1/1 [==============================] - 0s 18ms/step
[1.]
11.png is non malicious
1/1 [==============================] - 0s 19ms/step
[1.]
12.png is non malicious
1/1 [==============================] - 0s 20ms/step
[1.]
13.png is non malicious
1/1 [==============================] - 0s 18ms/step
[1.]
14.png is non malicious
1/1 [==============================] - 0s 22ms/step
[1.]
15.png is non malicious
1/1 [==============================] - 0s 18ms/step
[1.]
```

Fig. 6.8 Images generated from NeT2I being uploaded to the CNN3L


(a) An image from the NeT2I algorithm

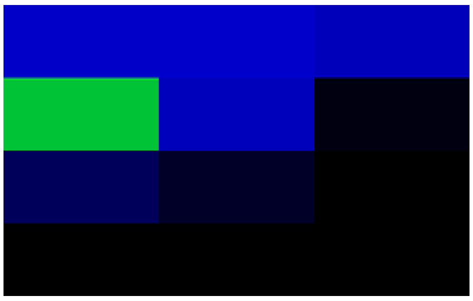

(b) A tiled image from the encoding algorithm of [1]

Fig. 6.9 A Visual Comparison of the Images for network traffic from the Testbed

Attack Type	Accuracy	Prediction	Recall	F1-Score
Port Scan	0.85	0.87	0.98	0.92
TCP SYN Flood	0.8	0.85	0.96	0.91
DDoS	0.74	0.78	0.83	0.81
Spoofed DDoS	0.74	0.78	0.83	0.81

Table 6.5 Confusion Matrix of the 5G-SiD in Stage-I Detection.

accuracy. Application of an anomaly detection algorithm may help in detecting attacks of unknown signatures, however, given the resource requirement and false alarm rate generated by such an algorithm, it is not viable to apply to a MEC architecture, since the nodes have low processing capabilities [214]. As our RTDL-NIDS can be extended into recognising more signatures by the introduction of additional rules, the application of a signature-based detection algorithm is considered a viable solution to be employed in the edge. As per [66], a quick detection with a lower accuracy at stage-I can be further improved with a higher accuracy at stage-II using an intelligent agent.

CNN3L Based Detection in Stage-II

Similarly, the confusion matrices for the two CNN3L models have been presented in Table 6.6 for the data collected in the 5G testbed. The employment of the proposed RTDL-NIDS, achieved a higher accuracy in comparison to the NIDS of [1]. The applied methodologies of detecting malicious traffic for the traffic from the testbed with the same 11 features as [1] has resulted in a higher detection rate of 97% from our proposed NIDS and 91% from the comparison NIDS respectively. We compared these results with those produced by other methodologies, for example, 47.19% in [203], binary classification in ML (linear regression (74.3%), Naive Bayes (77.3%), k Nearest Neighbour (81%), radial basis function in support vector machine (65.3%)) and an ensemble algorithm (97.2%) in [113] and deep neural network (80.1%) in [204] 37.15% in [205], 81.42% [206] and 72.57% Recall in [158].

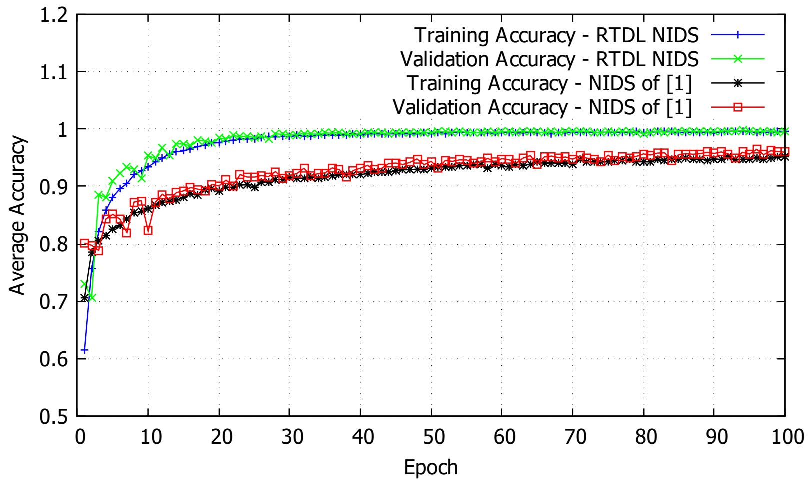
Algorithm	Accuracy	Prediction	Recall	F1-Score
Proposed IDS	0.97	0.96	0.97	0.96
NIDS of [1] + CNN3L	0.91	0.9	0.92	0.91

Table 6.6 Confusion Matrix of the CNN3L in Stage-II Detection.

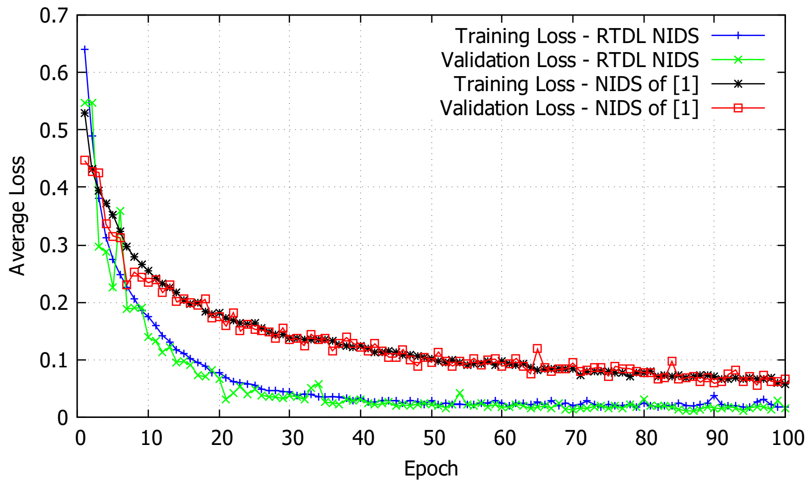
Results pertaining to accuracy during training and validation have been depicted in Fig. 6.10. By observing the training and validation accuracy (Fig. 6.10a) for both RTDL-NIDS and NIDS of [1], we can state that the CNN3L algorithm reached a steady state after epoch 70. Also, our model did not experience overfitting of data, as the validation accuracy curve didn't decrease. At epoch 70, acquired training and validation accuracy for the RTDL-NIDS were 0.992950022, 0.994000018, and for the NIDS of [1] acquired training and validation accuracy were 0.938750029, 0.942799995, respectively. Therefore, We can state that the RTDL-NIDS achieved a higher training and validation accuracy as opposed to NIDS of [1]. Similarly, Fig.6.10b presents the accumulated loss during training and validation. RTDL-NIDS recorded a lower loss compared to the NIDS of [1]. At epoch 70, the recorded loss for RTDL-NIDS during training and validation were 0.029572723, 0.021679578 and for the NIDS of [1], recorded loss during training and validation were 0.09036645, 0.101018724, respectively.

6.6.4 Comparison of Computational Complexity

We next present the computation complexity for the two NIDS, as depicted in Fig. 6.5. As the number of images being generated is arbitrary since it is based on the volume of real-time network traffic present in the network, we present the computation complexity with respect to the observed throughput in the network. We initially present the results for the two signature detection mechanisms, 5G-SiD, and 5G-SiD+Cyhper, followed by the computational complexity for the NeT2I and encoding of [1], when applied in the 5G-MEC mobile telecommunication testbed, and lastly, we present the computational complexity, as a whole, for the new RTDL-NIDS and the NIDS of [1]. Results pertaining to 5G-SiD and 5G-SiD+Cypher have been graphically depicted in Figs.6.11. Results collected from the NeT2I and encoding of [1] employed in the 5G-MEC mobile telecommunication testbed can be found in Figs.6.11 and Fig.6.12.

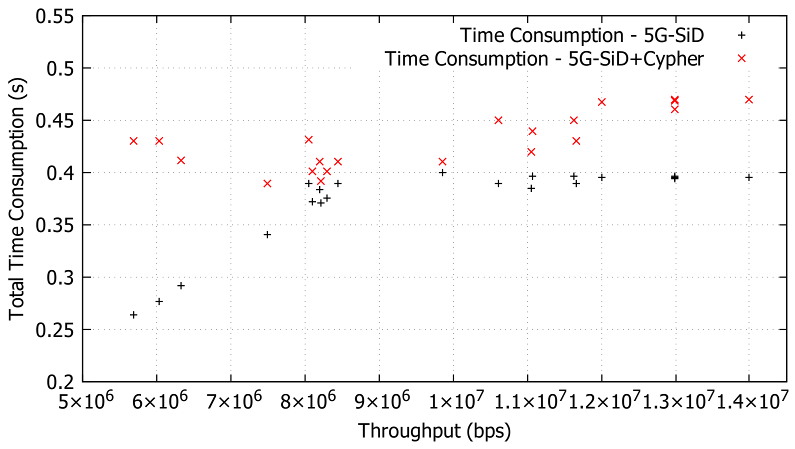


(a) Training and Validation Accuracy

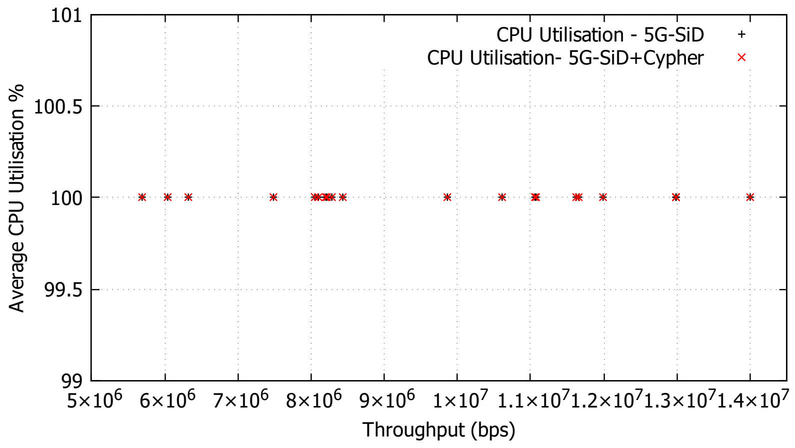


(b) Training and Validation Loss

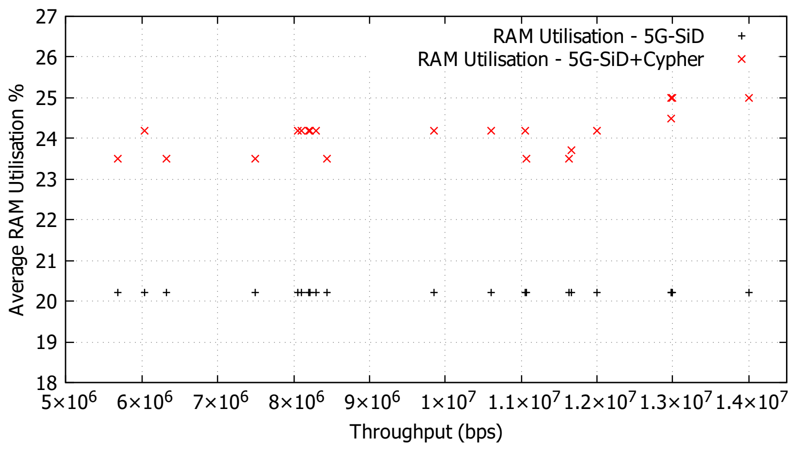
Fig. 6.10 Training and Validation Data from the Testbed



(a) Time consumption for 5G-SiD and 5G-SiD+Cypher as per [1]



(b) CPU computational complexity for 5G-SiD and 5G-SiD+Cypher as per [1]



(c) RAM computational complexity for 5G-SiD and 5G-SiD+Cypher as per [1]

Fig. 6.11 Computational Complexity for 5G-SiD and 5G-SiD+Cypher Algorithm

Computational Complexity for 5G-SiD and 5G-SiD+Cypher

- **Time Consumption**

  Fig.6.11a, depicts in black the time consumption of the 5G-SiD and in red depicts the time consumption of the 5G-SiD+Cypher. We employed the time spent for each algorithm against the throughput. A similar approach can be seen in the works of [92]. The 5G-SiD consumed an average of 0.369s to collect network traffic, segregate, label, calculate variations, check for flags and return a CSV with network traffic that corresponds to signature detection with an average throughput of $10*10^6$ bps. For a similarly recorded throughput, the 5G-SiD+Cypher recorded 0.428s. Irrespective of the recorded throughput in the network, the 5G-SiD+Cypher algorithm has consumed more time in signature detection and in preparing a CSV file for NeT2I as opposed to 5G-SiD. This can be observed when the recorded throughput was at $6.5*10^6$ bps, a noticeable gap between the two algorithms is visible. Hence, we can state that the employment of the cypher algorithm consumed an additional 14% of time as opposed to 5G-SiD.

  In Fig.6.11a,we can notice a higher time consumption to a low throughput for the 5G-SiD+Cypher, denoted in red, during the initial distribution of time. The higher time consumption can be regarded as due to the high processing requirement and the time consumption of the 5G-SiD+Cypher algorithm. Collated traffic features such as MAC addresses and IP addresses must be translated into a numerical representation. This process contributes towards the processing time. A similar increase of time to a lower throughput has also been recorded by the work presented by [92] when the network speed grows from 0 bps, and then suddenly grows with the increment of throughput.

- **CPU Utilisation**

  Fig.6.11b, depicts in black the CPU utilisation of the 5G-SiD and in red depicts the CPU utilisation of the 5G-SiD+Cypher. 5G-SiD and 5G-SiD+Cypher utilised 100% of the CPU resources whilst being applied to the 5G-MEC mobile telecommunication testbed. As both 5G-SiD and 5G-SiD+Cypher are characterized as Aperiodic tasks and therefore acquiring the full resources of the CPU, is acceptable.

- **RAM Utilisation**

  Fig.6.11c, depicts in black the RAM utilisation of the 5G-SiD and in red depicts the RAM utilisation of the 5G-SiD+Cypher. 5G-SiD utilised 20.2% of RAM for the duration of the execution whilst the 5G-SiD+Cypher algorithm utilised 24.3% of

RAM for an averaged throughput of $10{*}10^6$ bps, signifying a 16% reduction of RAM utilisation in 5G-SiD as opposed to 5G-SiD+Cypher.

Computational Complexity for NeT2I and Encoding of [1]

- **Time Consumption**

  Fig.6.12a present the results collected from the NeT2I and the encoding of [1] when applied to real-time network traffic, as opposed to when the aforementioned algorithms were applied in a desk approach (to a dataset). On average, the NeT2I algorithm consumed 7.1s on average to translate a CSV file containing network traffic to images, whereas the encoding algorithm of [1], consumed an average of 30.786s. The creation of variable x and y coordinates have contributed towards a 76% increment in time for the encoding algorithm of [1] as opposed to NeT2I. Thus confirming the results obtained from the aforementioned two algorithms when applied in a desk approach [215] (See Chapter 5.4.2).

- **CPU Utilisation**

  Fig.6.12b present the computational complexity for the NeT2I and encoding of [1] when applied for the creation of images in the 5G-MEC Mobile Telecommunication testbed. Both NeT2I and the encoding of [1] recorded a CPU utilisation of 100%, by being an Aperiodic Task. NeT2I and encoding of [1] results pertaining to CPU utilisation in this iteration are also in line with the results when applied in the desk approach (See Chapter 5.4.2).

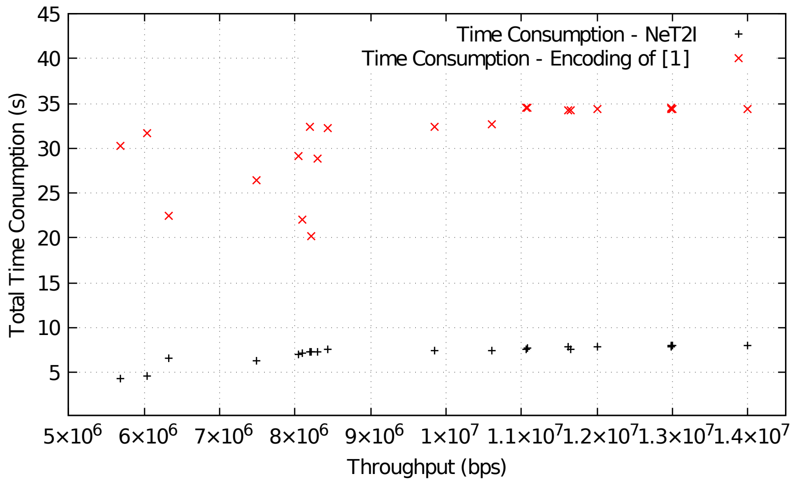
- **RAM Utilisation**

  Fig.6.12c presents the results of RAM utilisation of NeT2I and encoding of [1] when applied to the real-time 5G-MEC mobile telecommunication testbed. NeT2I utilised 21.3% of the available RAM when applied to the real-time testbed. Encoding of [1] utilised 28.3% of available RAM. NeT2I used 24% less RAM in the real-time testbed as opposed to the encoding of [1] NeT2I and encoding of [1] results pertaining to RAM utilisation in this iteration are also in line with the results when applied in the desk approach (See Chapter 5.4.2).

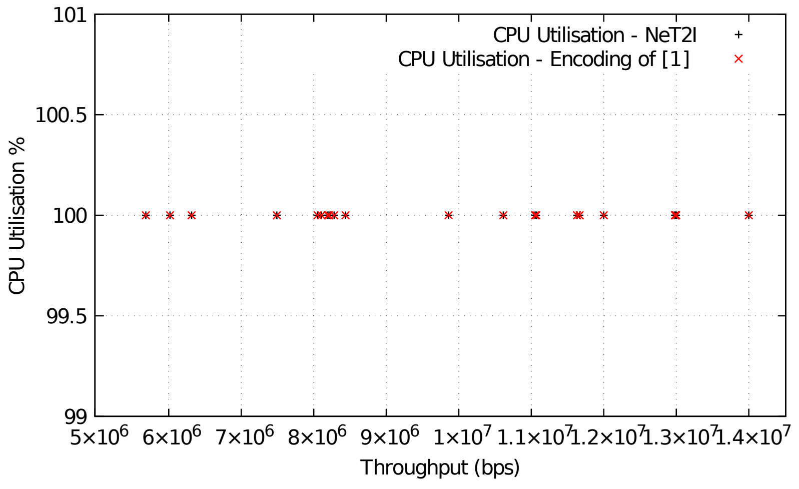
Computational Complexity for RTDL-NIDS and NIDS of [1]

- **Time Consumption**

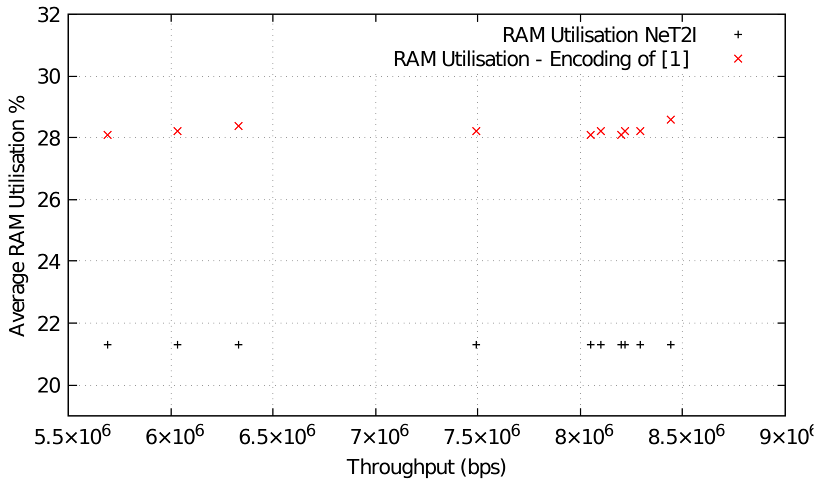
  Fig.6.13a presents the cumulative results of the two NIDSs pertaining to time consumption. On average the new RTDL-NIDS (black) spent 7.5s for signature



(a) Time consumption for NeT2I and encoding of [1]

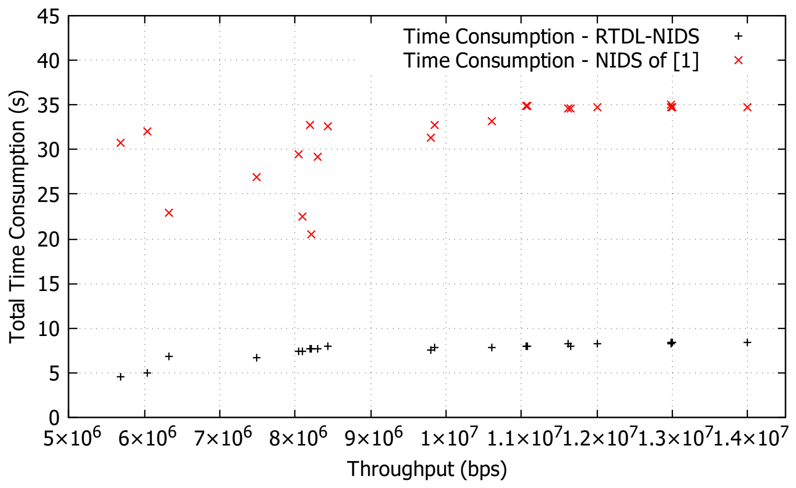


(b) CPU computational complexity for NeT2I and encoding of [1]

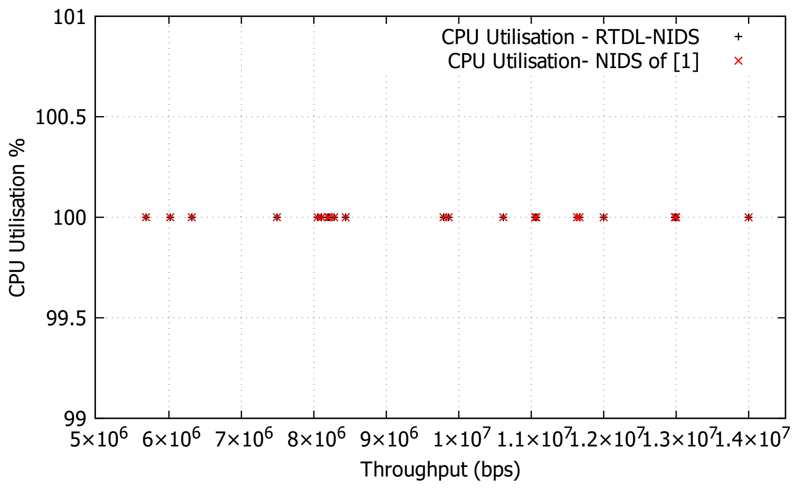


(c) RAM computational complexity for NeT2I and encoding of [1]

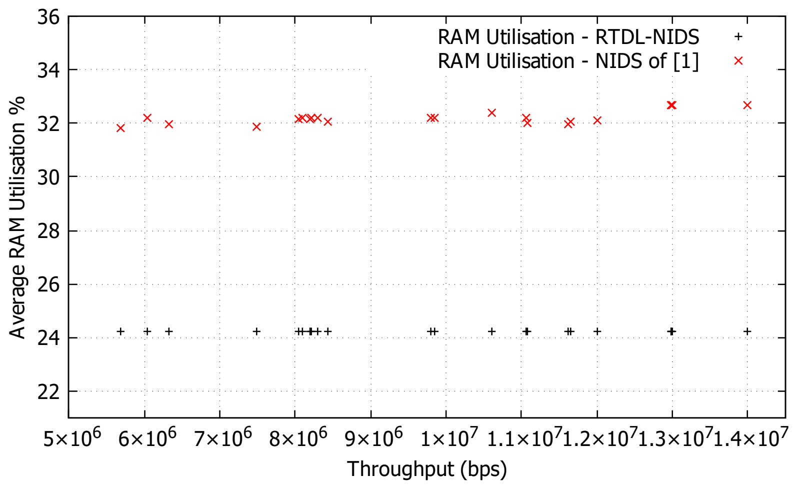
Fig. 6.12 Computational Complexity for NeT2I and encoding of [1]



(a) Time consumption for RTDL-NIDS and the NIDS of [1]



(b) CPU computational complexity for RTDL-NIDS and NIDS of [1]



(c) RAM computational complexity for RTDL-NIDS and NIDS of [1]

Fig. 6.13 Computational Complexity for Proposed NIDS and NIDS of [1]

detection and image creation, whereas the NIDS of [1] (red) consumed 31s to conduct the same, highlighting a reduction of time consumption by 75%.

- **CPU Utilisation**

  Fig.6.13b presents the computational complexity pertaining to the CPU utilisation in the RTDL-NIDS and the NIDS of [1]. Similar to the previous findings (5G-SiD, 5G-SiD+Cypher, NeT2I, and encoding of [1]) pertaining to the CPU utilisation, the execution of RTDL-NIDS and the NIDS of [1] recorded a CPU utilisation of 100%. The usage of 100% is in line with the characteristics of an Aperiodic Task.

- **RAM Utilisation**

  The RAM utilisation for the new RTDL-NIDS and the NIDS of [1], have been presented in Fig. 6.13c. On average the new NIDS (black) consumed 24.25% for signature detection and image creation, whereas the NIDS of [1] (red) consumed 32.19% to conduct the same, highlighting a reduction of RAM utilisation by 24%.

As per the findings, the application of the proposed new RTDL-NIDS has utilised a lower computational complexity as opposed to the NIDS of [1]. The data suggest that the application of the new RTDL-NIDS is the most viable approach to detect malicious activity in the MEC architecture as a form of MEC-IDS. Although both underpinned algorithms utilised 100% of the CPU resources, new RTDL-NIDS executed for a shorter time period. With the additional requirements of cyphering the IP addresses and the creation of x and y coordinates in the encoding phase, have acquired more time to compute. This also resulted in a higher requirement for RAM resources. From the collated data, RTDL-NIDS require 500MB of RAM and 1 CPU to conduct signature detection, encoding to images, and uploading to the CNN3L algorithm. Thus highlighting the suitability and the applicability of the proposed RTDL-NIDS to a MEC-IDS node, capable of detecting malicious attacks.

6.7 Chapter Summary

In this chapter, we employed the encoding mechanisms and the detection algorithm that we presented and evaluated in [215] on the testbed [152] and collected results in real-time. We conducted this research to advance the security of the 5G-MEC infrastructure by employing an effective IDS based on CNN with low resource utilisation, and the employment of a real-time testbed provided us with many conclusive results. In the next chapter, we provide the conclusion, present our future work, and close this thesis.

Chapter 7

Conclusion

7.1 Overview

The aim of this research was to develop an efficient 5G mobile telecommunication testbed with MEC architecture and protect these low-resource bound nodes from malicious attacks using an intelligent agent. This chapter concludes this research by reviewing the findings, discussions, contributions, and limitations and discussing areas for, future research, that follow on from the research that we have carried out.

7.2 Research Findings

The primary research of this thesis was based on what steps can be taken to enhance the security of 5G-MEC against malicious attacks in a 5G-MEC network, by utilising a DL agent. A thorough study of the existing literature highlighted research gaps that motivated our research for this thesis. They can be found in Chapters 1 and 2. Underlying research questions derived from the primary research question can be found in Table 7.1. Having low computational power makes the MEC nodes vulnerable to attack since IDS/IPS systems inherently require complex computational power [112]. The utilisation of a highly resource-consuming algorithm or software launched in a MEC node also makes a barrier to providing a consistent service to the users [216], since MEC nodes are inherently short-lived with low resource availability. Attacks should be detected swiftly, effectively, and efficiently, to provide a service as free from degradation due to attacks, as possible. To facilitate this, a network with faster convergence of network packets with reduced redundancies, jitter, loss, and delays in the core network would be beneficial. Furthermore, the detection should

Primary Research Question: What steps can be taken to enhance the security of 5G-MEC against malicious attacks in a 5G network, by utilising a DL agent?
Sub Questions
RQ1:How does programmability in both the control plane and data plane affect network performance?
RQ2: How do we design and build a testbed for 5G mobile telecommunications?
RQ3: How can we design new algorithms for intrusion detection using a DL agent in 5G-MEC?
RQ4: How do we apply DL in real-time to intrusion detection in 5G-MEC mobile telecommunication testbed?

Table 7.1 Underlying Research Questions derived from the Primary Research Question

occur in an effective and efficient manner in real-time on a 5G testbed. For this, the newly developed algorithms were implemented on the 5G testbed for a real-time detection.

Discussions based on our findings were organised covering the four main research questions and objectives. They are 1. to improve the performance of the core, 2. To design and develop a 5G-MEC Mobile Telecommunication Testbed, 3. To encode, and decode network traffic effectively and efficiently with low computational complexity, 4. And finally, to conduct a real-time detection based on Deep Learning in 5G-MEC Mobile Telecommunication Testbed. Experiments were conducted for each research question as per Table 7.2 and the findings are as follows.

- **RQ1: How does programmability in both the control plane and data plane affect network performance?**

  The first objective was to improve the performance of the core network by increasing throughput, bps, and QoS by reducing delay, jitter, packet loss and latency. To achieve the envision of 5G and beyond, programmability at both the control plane (*SDN*) and the data plane (*P4*) is crucial when providing resilient and high-performing service to various use cases. The potential to pass packets in parallel as opposed to sequentially (*SDN+OvS*), establishes *SDN+P4* as an advantage in providing an improved platform. The capability to parse packets without modification or an upgrade to the parser, offers flexibility to service providers when introducing new services and packet headers.

  The emulations of *SDN+P4* successfully outperformed *SDN+OvS* with respect to the topologies, traffic types and network matrices that follow. The implementation

Research Question 1: Evaluation of Control Plane and Data Plane Programmability	Research Question 2: Implementation of a 5G Mobile Telecommunication Testbeds	Research Question 3: New Algorithms for the Detection of Malicious Network Traffic	Research Question 4: Real-time Implementation of Intrusion Detection in the 5G Mobile Telecommunication Testbed
1. Emulation of Control Plane and Data Plane Programmability	1. Study of the existing simulators and emulators	1. Study of the existing Machine Learning and Deep Learning Algorithms	1. Study of the existing hypothetical two-staged IDSs
2. Emulation of Different Topologes	2. Implement a 5G Mobile Telecommunication Testbed	2. Study of the existing methods of data encoding	2. Study of network traffic collection techniques
3. Emulation of Various Network Traffic	3. Implement programmability in the 5G Mobile Telecommunication Testbed	3. Implement algorithms for data encoding and decoding	3. Implement a signature detection algorithm
4. Analysis of Key Performance Indicators	4. Emulate a MEC network	4. Implement a detection algorithm by employing CNN	4. Incorporate encoding algorithm after signature detection
	5. Emulate Network traffic between User and MEC edge node	5. Evaluate existing publicly available datasets	5. Evaluate the computational complexity of the encoding algorithm against literature
	6. Conduct Malicious Network Attacks	6. Encode data from the Datasets using the developed encoding algorithm	6. Link Google Colab
	7. Analysis of Key Performance Indicators	7. Evaluate the computational complexity of the encoding algorithm against literature	7. Apply the Confusion Matrix
	8. Creation of Datasets from 5G-MEC network traffic	8. Train and Test the CNN algorithm	
		9. Apply the Confusion Matrix	

Table 7.2 Investigation drawn from each Research Question

of *SDN+P4* and *SDN+OvS* was tested across multiple topologies: multi-path, grid and the Internet. In each topology, network traffic such as ICMP, TCP, UDP, and CDN was instantiated. Performance matrices such as throughput, bps, delay, packet loss, jitter and latency were collected across each topology with respect to individual network traffic and, lastly in a simultaneous execution of all network traffic. The collected results accentuated the performance gain achieved via the employment of *SDN+P4*. The results also highlighted how the sequential processing of *OvS*, has created a drop in consistent throughput in the performance for *SDN+OvS*. The collected results also highlighted the advantage of parallel processing with *SDN+P4*, in similar circumstances to those in which *SDN+OvS*, failed to provide higher throughput. Lastly, our research suggests that the application of *SDN+P4* serves as a viable option to be considered in reducing redundancy and delays, that exist in the core network, and which could potentially disrupt achieving KPIs for 5G and beyond. The emulations and results were presented and discussed thoroughly in Chapter 3.4.

- **RQ2: How do we design and build a testbed for 5G mobile telecommunications?**

  The second objective was to develop a 5G-MEC mobile telecommunication testbed to highlight the significance of employing a 5G testbed and a dataset for research based on 5G. The collection of network traffic, both malicious and non_malicious in the 5G testbed aids us in developing datasets, that were used in comparison to publicly available but inappropriate datasets that have been employed for 5G research in the literature.

For the purpose of this research, we evaluated the UNSW NB-15 and InSDN datasets against the dataset collected from our testbed. The collected dataset differed from the UNSW NB-15 and InSDN datasets as it contained the 5G traffic types and protocols, not found in the UNSW NB15 and InSDN datasets. They were, SCTP, S1AP, SSDP, GTP, eNB and MME traffic and eNB and USRP traffic. We also included malicious network traffic such as DoS, DDoS, TCP Syn Flood, Botnet Traffic, and Port Scanning. The collection of the aforementioned network traffic accentuated and provided a certitude to this research objective. The development, deployment and the collection of 5G datasets have been discussed in Chapter 4.3.

- **RQ3: How can we design new algorithms for intrusion detection using a DL agent in 5G-MEC?**

  The third objective was to evaluate the existing ML/DL algorithms and to develop an ideal algorithm that could be applied to the 5G-MEC mobile telecommunication testbed, for the detection of malicious network traffic. This objective also carried a sub-objective of feature selection and data preprocessing. Upon identifying the advantages and the performance gain of CNN, an algorithm capable of binary classification was developed. A significant challenge to the application of CNN involved data encoding. To this end, the NeT2I algorithm was developed. Although it created an elegant solution for the encoding process, the algorithm had to be evaluated. See Chapter 5. Both algorithms were evaluated based on computational complexity and the acquired accuracy from the detection algorithm (CNN3L). As these algorithms would eventually be implemented in an MEC node, the computational complexity must be minimal. NeT2I recorded a lower computational complexity as opposed to the comparative algorithm. This was the case for network traffic based on UNSW NB-15, InSDN and the collected 5G dataset. Upon successful encoding, the detection algorithm, CNN3L, successfully detected malicious traffic with 96% accuracy for the images derived from NeT2I, as this algorithm, didn't employ a Cipher mechanism. To emulate the functions and capabilities of the MEC, the CNN3L algorithm was launched in Google Colab.

  Lastly, the detected images were decoded in order to collect malicious user information. To this end, I2NeT was developed. To compare the performance of the I2NeT, a comparative decoding algorithm was also developed. Across, UNSW NB-15, InSDN, and the collected 5G dataset, I2NeT showed significantly less computational complexity, highlighting that both NeT2I and I2NeT can be implemented in an MEC node. The development and evaluation of the new algorithms have been presented and discussed in Chapter 5.4.

- **RQ4: How do we apply DL in real-time to intrusion detection in 5G-MEC mobile telecommunication testbed?**

  The last objective was to incorporate the developed algorithms from the previous objective into the 5G-MEC mobile telecommunication testbed.

  To overcome the challenges, the following were developed and applied to the testbed. 5G-SiD algorithm was developed to conduct a quick signature detection. Following the 5G-SiD, NeT2I, Fuse_Google_Drive, CNN3L, and I2NeT were developed as sub-routines. The 5G-SiD was found to detect malicious attacks with an accuracy of 82%. As this also labels suspicious traffic as malicious, it is followed by a more accurate detection by the CNN3L, however, the lower accuracy achieved was negligible. At the end of the signature-based detection, NeT2I gets executed based on the CSV file. The generation of images using NeT2I is computationally more viable than other popular NIDS algorithms. Generated images were uploaded in real-time to CNN3L, which was launched in Google Colab.

  Given that this is the first image-based NIDS, launched in real-time, the recorded accuracy along with the computational complexity provides eligibility to be implemented in a real MEC node as MEC-IDS. Our findings at the end of these last four objectives show that a real-time detection based on DL is achievable, as a CNN3L algorithm can be implemented in a 5G-MEC mobile telecommunication testbed; as a two-staged detection which conducts, both a quick detection based on known signatures, followed by a more accurate DL based detection. The development, and the application of RTDL-NIDS and the evaluation of it has been presented in Chapter 6.6.

7.3 Contribution to Knowledge

Significant and original contributions to knowledge can be listed as follows. Fig 7.1 depicts the contributions from this thesis in the overview. Overall, the thesis initially evaluated the performance of *SDN+OvS* against *SDN+P4*, developed a new 5G-MEC mobile telecommunication testbed, developed and evaluated new algorithms for the detection of

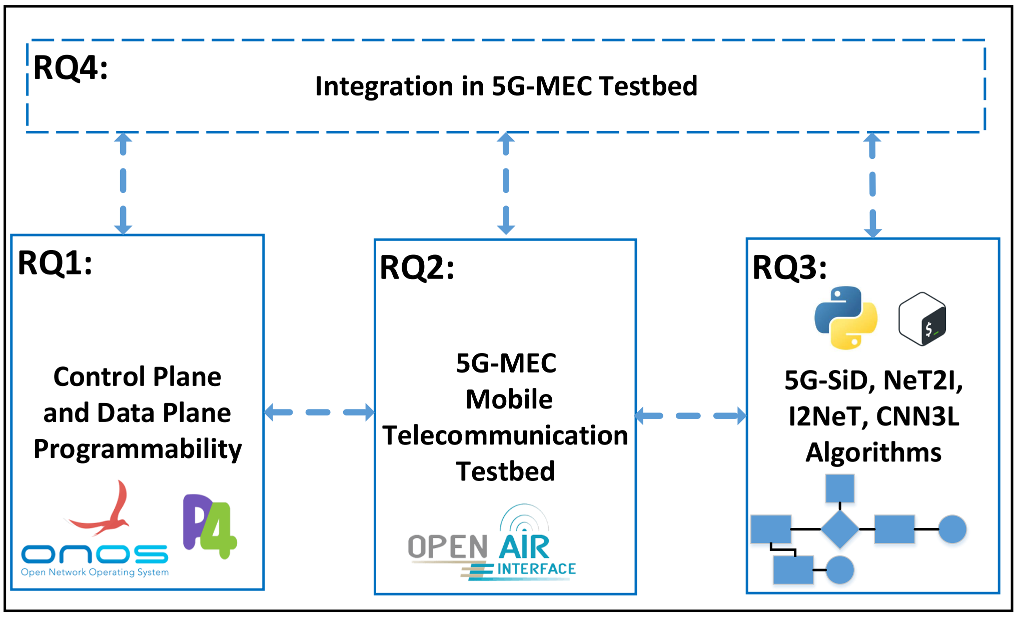

RQ4:
Integration in 5G-MEC Testbed
RQ1:
Control Plane and Data Plane Programmability
ONOS Open Network Operating System
P4
RQ2:
5G-MEC Mobile Telecommunication Testbed
OPEN AIR INTERFACE
RQ3:
5G-SiD, NeT2I, I2NeT, CNN3L Algorithms

Fig. 7.1 Contributions from this thesis

malicious traffic in 5G-MEC, proposed and developed a new RTDL-NIDS, a Real-Time Deep Learning based Network Intrusion Detection System.

- **Development of Real-Time Deep Learning based Network Intrusion Detection System (RTDL-NIDS):** A novel RTDL-NIDS which encompasses a 5G-SiD algorithm capable of conducting a quick signature detection, employment of NeT2I, CNN3L, and I2NeT was developed to conduct intrusion detection in real-time in the developed 5G-MEC Mobile Telecommunication testbed. The RTDL-NIDS was evaluated against the real-time application of the NIDS presented by [1]. The current literature has been aimed at applying an intelligent agent (ML or DL) towards the detection of malicious traffic in an offline mode (desk approach). In this thesis, the application of intrusion detection using an intelligent agent has been carried out in real-time as opposed to the desk approach.

- **Development of New Algorithms for the Detection of Malicious Traffic in 5G-MEC:** The development of NeT2I for encoding network traffic, including IP addresses and MAC address to RGB colour space, a new algorithm (I2NeT) to decode RGB images back to network traffic, and finally a new algorithm (CNN3L) for the detection of malicious network traffic. The developed algorithms were evaluated against a similar encoding, decoding, and detection algorithm [1], based on computational complexity and generated accuracy. Current literature uses a Cypher algorithm or evades the encoding of network information such as IP addresses and MAC addresses in the RGB colour space as IP addresses and MAC addresses which exceed 24 bits. Information on the significance of NeT2I has been discussed in Chapter 5. By using NeT2I, we present a new algorithm to encode and decode network information to RGB colour space, to generate images that can be utilised for intrusion detection. The development of a decoding algorithm has not been attempted in the current literature, whereas I2NeT, was developed to introduce the importance of a decoding algorithm that can be utilised towards future intrusion prevention problems. The developed algorithms (NeT2I and I2NeT) were also evaluated based on computational complexity that has not been attempted in the current literature.

- **A Development of a 5G-MEC Mobile Telecommunication Testbed with P4 and Dataset:** A 5G-MEC Mobile Telecommunication testbed was developed and launched, employing P4 BMv2 switches, for the creation of 5G datasets. The acquired 5G datasets were evaluated against publicly available datasets. The current literature conducts security-based research for 5G, by employing datasets that were generated on LAN networks. The network traffic on a LAN network differs significantly from that found in a 5G-MEC network. The created dataset and the evaluation of it against popular datasets (UNSW NB-15 and InSDN) highlight the need to employ a relatable dataset to 5G and beyond intrusion detection research.
- **An Evaluation of *SDN+P4* against *SDN+OvS*:** Across multiple topologies and by employing various network traffic, emulations were conducted to evaluate the performance of *SDN+P4* against *SDN+OvS*. Our research established that with the initialisation of *SDN+P4* with parallel processing of packets, various applications have better performance in comparison to applications run over *SDN+OvS*. The current literature has employed P4 to improve the performance of a particular case study or an application. In this thesis, we applied *SDN+P4* to improve the performance of the network irrespective of the type of traffic and the topology.

7.3.1 Research Significance

The significance of this research is that it provides a mechanism to conduct a multi-staged detection using a quick signature-based detection scheme and a more accurate DL-based detection in a real-time 5G-MEC mobile telecommunication testbed, using RTDL-NIDS. The developed algorithms have also been evaluated computationally so that they can be implemented in a low-processing power environment.

This research demonstrated the importance of applying data plane programmability to faster processing of network packets. This was corroborated by the emulation results and the faster processing achieved by the edge switches in the 5G-MEC mobile telecommunication testbed. Having programmability in the data plane (P4) will also aid in providing specific functions on top of connectivity. Faster processing of network packets will also aid in translating in-directly toward faster detection of malicious network traffic. Having utilised data plane programmability at the edge of the network will contribute more to both the end user and the service provider in a more meaningful manner in addition to providing connectivity.

Another fact that this research highlighted was that the application of a testbed to imitate a 5G mobile telecommunication testbed provided more meaningful and contributory results as opposed to applying an algorithm to a dataset collected on a LAN or a campus network. As the current research for securing 5G was carried out using training/testing an algorithm using a publicly available dataset, the application of a 5G mobile telecommunication testbed, highlights traffic patterns and protocols that cannot be obtained using LAN or campus networks. Our evaluative results highlight the unique traffic patterns in a 5G mobile network.

Following the development of the 5G-MEC mobile telecommunication testbed, this research highlights the importance of developing encoding and decoding algorithms with minimal computational complexity. NeT2I, I2NeT, and 5G-SiD were developed with respect to minimal computational complexity. Not only do these algorithms outperform the comparative algorithms in terms of computational complexity, but they also aided in producing a higher detection rate when the images were applied to CNN3L. These results point to the fact that these algorithms can be implemented in a MEC node with low computational power.

This research also signifies that the implementation of DL can be achieved in real-time with the employment of a 5G-MEC mobile telecommunication testbed. The application of the aforementioned algorithms can be implemented in real-time, thus eliminating the desk approach of applying DL to a dataset to test the effectiveness of classification. Given that the CNN3L was launched on a Google Colab environment, the detection algorithm can be easily launched on-the-fly as an MEC-IDS node.

7.4 Future Work

This thesis provides an early investigation into real-time intrusion detection using DL in 5G-MEC followed by an implementation. Future work identified from the research presented in this thesis includes:

- Publication of further results from the employment of NeT2I, I2NeT, and CNN3L across other publicly available datasets. This will help to further evaluate the effectiveness of the developed NeT2I, I2NeT, and CNN3L algorithms.
- Improvements to the 5G-MEC mobile telecommunication testbed in, for example, the extension of the RAN network by employing multiple USRP devices.
  - This will extend the RAN network and nodes can experience mobility whilst being connected to the network. Research such as mobility prediction using ML/DL can be further investigated by employing this 5G-MEC mobile telecommunication testbed, by improving the collected 5G datasets.
  - Employing two physical nodes to independently initialise the RAN network and the Core network. This will further improve the performance of the network and the algorithms employed.
  - The employment of a Tofino P4 [217] switch will further improve the performance of the network [218] and will aid in the faster detection of malicious network traffic as opposed to the P4 switch, implemented on the NIC.
- RTDL-NIDS can be further improved. Currently, the algorithm is executed using a single thread. This can be further improved by employing multiple threads to execute this algorithm: i.e 5G-SiD can be implemented on a different thread to NeT2I and Fuse_Google_Drive, and finally, I2NeT on another thread. This will improve the detection even further. Although the current execution time for NeT2I is smaller, removing this execution time from the main thread, where signature detection occurs, should provide improved detection and computational complexity.
- Current RTDL-NIDS can successfully detect malicious network traffic. But at present, it doesn't prevent it. Upon successful completion of another API connecting CNN3L with I2NeT, information such as the IP address and the MAC address of the advisories can be passed to the P4 switch to prevent access [190]. In this manner, a successful, IDS and an IPS based on DL can be implemented in the 5G-MEC mobile telecommunication testbed.
- As we collected computational complexity in our algorithms, we can extend this work in order to determine the baseline performance of the 5G-MEC nodes. By doing so, we can conduct anomaly detection using the CNN3L algorithm. As the detection of unknown attacks seems difficult with signature detection mechanisms, the employment

of an anomaly detection system seems promising. We can include computational complexity as a feature of the NeT2I and train the CNN3L to detect anomalies along with known intrusions.

- The current MEC-IDS can be extended to listen to multiple nodes in the MEC network. This extension will provide a mechanism to detect malicious traffic in the entire 5G-MEC network.
- As the 5G-MEC mobile telecommunication testbed grows with multiple RAN nodes, Federated learning [219] can be implemented to train and test a DL algorithm for IDS and IPS in a decentralised manner. Since multiple MEC nodes can be launched in the improved testbed, multiple MEC-IDS nodes can collect network traffic, and train a DL algorithm for improved detection and prevention of malicious access.

7.5 Closing

Detection of malicious network traffic aimed towards the edge using DL implemented in real-time employing a 5G-MEC mobile telecommunication testbed has been presented in this thesis. This implementation paves the way for researchers to consider real-time approaches and new 5G datasets as opposed to simulators/emulators or desk approaches for security-based research in 5G and beyond. With the implementation of future work and further technological improvements, this research can be truly transcended into a real-world solution that can be launched on-the-fly in 5G and beyond.

References

[1] Ahmed H Janabi, Triantafyllos Kanakis, and Mark Johnson. Convolutional neural network based algorithm for early warning proactive system security in software defined networks. *IEEE Access*, 10:14301–14310, 2022.

[2] Sami Kekki, Walter Featherstone, Yonggang Fang, Pekka Kuure, Alice Li, Anurag Ranjan, Debashish Purkayastha, Feng Jiangping, Danny Frydman, Gianluca Verin, et al. Mec in 5g networks. *ETSI white paper*, 28:1–28, 2018.

[3] Belal Ali, Mark A Gregory, and Shuo Li. Multi-access edge computing architecture, data security and privacy: A review. *IEEE Access*, 9:18706–18721, 2021.

[4] Nathan Shone, Tran Nguyen Ngoc, Vu Dinh Phai, and Qi Shi. A deep learning approach to network intrusion detection. *IEEE transactions on emerging topics in computational intelligence*, 2(1):41–50, 2018.

[5] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, 5:21954–21961, 2017.

[6] Abdullah J Alzahrani and Ali A Ghorbani. Real-time signature-based detection approach for sms botnet. In *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, pages 157–164. IEEE, 2015.

[7] Lorenzo Fernández Maimó, Ángel Luis Perales Gómez, Félix J García Clemente, Manuel Gil Pérez, and Gregorio Martínez Pérez. A self-adaptive deep learning-based system for anomaly detection in 5g networks. *Ieee Access*, 6:7700–7712, 2018.

[8] Peter J Denning. Is computer science science? *Communications of the ACM*, 48(4): 27–31, 2005.

[9] Walter F Tichy. Should computer scientists experiment more? *Computer*, 31(5): 32–40, 1998.

[10] Matti Tedre and Nella Moisseinen. Experiments in computing: A survey. *The Scientific World Journal*, 2014, 2014.

[11] Keith Kirkpatrick. Software-defined networking. *Communications of the ACM*, 56(9): 16–19, 2013.

[12] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[13] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2): 69–74, 2008.

[14] Anders Nygren, B Pfaff, B Lantz, B Heller, C Barker, C Beckmann, D Cohn, D Malek, D Talayco, D Erickson, et al. Openflow switch specification version 1.5. 1. *Open Networking Foundation, Tech. Rep*, 2015.

[15] Pedro Heleno Isolani, Juliano Araujo Wickboldt, Cristiano Bonato Both, Juergen Rochol, and Lisandro Zambenedetti Granville. Interactive monitoring, visualization, and configuration of openflow-based sdn. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 207–215. IEEE, 2015.

[16] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[17] Anirudh Sivaraman, Changhoon Kim, Ramkumar Krishnamoorthy, Advait Dixit, and Mihai Budiu. Dc. p4: Programming the forwarding plane of a data-center switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, pages 1–8, 2015.

[18] Cisco Visual Networking Index. Forecast and methodology, 2016–2021. *White Paper, June*, 2017.

[19] V Cisco. The zettabyte era: trends and analysis. updated (07/06/2017), 2017.

[20] Donato Emma, Salvatore Loreto, Antonio Pescapé, and Giorgio Ventre. Measuring sctp throughput and jitter over heterogeneous networks. In *20th International Conference on Advanced Information Networking and Applications-Volume 1 (AINA'06)*, volume 2, pages 5–pp. IEEE, 2006.

[21] Winarno Sugeng, Jazi Eko Istiyanto, Khabib Mustofa, and Ahmad Ashari. The impact of qos changes towards network performance. *International Journal of Computer Networks and Communications Security*, 3(2):48–53, 2015.

[22] Jose F Monserrat, Genevieve Mange, Volker Braun, Hugo Tullberg, Gerd Zimmermann, and Ömer Bulakci. Metis research advances towards the 5g mobile and wireless system definition. *EURASIP Journal on Wireless Communications and Networking*, 2015(1):53, 2015.

[23] Victor Boteanu, Hanieh Bagheri, and Martin Pels. Minimizing arp traffic in the ams-ix switching platform using openflow. *Cited on*, page 7, 2013.

[24] Danilo Cerović, Valentin Del Piccolo, Ahmed Amamou, Kamel Haddadou, and Guy Pujolle. Fast packet processing: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):3645–3676, 2018.

[25] Athanasios Liatifis, Panagiotis Sarigiannidis, Vasileios Argyriou, and Thomas Lagkas. Advancing sdn from openflow to p4: A survey. *ACM Computing Surveys*, 55(9):1–37, 2023.

[26] Naveen Kr Sharma, Antoine Kaufmann, Thomas Anderson, Arvind Krishnamurthy, Jacob Nelson, and Simon Peter. Evaluating the power of flexible packet processing for network resource allocation. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 67–82, 2017.

[27] Jeferson Santiago da Silva, François-Raymond Boyer, Laurent-Olivier Chiquette, and JM Pierre Langlois. Extern objects in p4: an rohc header compression scheme case study. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 517–522. IEEE, 2018.

[28] Han Wang, Robert Soulé, Huynh Tu Dang, Ki Suh Lee, Vishal Shrivastav, Nate Foster, and Hakim Weatherspoon. P4fpga: A rapid prototyping framework for p4. In *Proceedings of the Symposium on SDN Research*, pages 122–135, 2017.

[29] Xiang Chen, Dong Zhang, Xiaojun Wang, Kai Zhu, and Haifeng Zhou. P4sc: Towards high-performance service function chain implementation on the p4-capable device. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1–9. IEEE, 2019.

[30] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, and Jennifer Rexford. Pisces: A programmable, protocol-independent software switch. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 525–538, 2016.

[31] P Gyanesh Kumar Patra, Fabricio E Rodriguez Cesen, Juan Sebastian Mejia, Daniel Lazkani Feferman, Levente Csikor, Christian Esteve Rothenberg, and Gergely Pongracz. Toward a sweet spot of data plane programmability, portability, and performance: On the scalability of multi-architecture p4 pipelines. *IEEE Journal on Selected Areas in Communications*, 36(12):2603–2611, 2018.

[32] Ting-Shan Wong and Steven SW Lee. Design of an in-band control plane for automatic bootstrapping and fast failure recovery in p4 networks. *IEEE Transactions on Network and Service Management*, 2023.

[33] K Tolga Bagci and A Murat Tekalp. Sdn-enabled distributed open exchange: Dynamic qos-path optimization in multi-operator services. *Computer Networks*, 162:106845, 2019.

[34] Kenneth L Calvert, Matthew B Doar, and Ellen W Zegura. Modeling internet topology. *IEEE Communications magazine*, 35(6):160–163, 1997.

[35] Vasileios Kotronis, Adrian Gämperli, and Xenofontas Dimitropoulos. Routing centralization across domains via sdn: A model and emulation framework for bgp evolution. *Computer Networks*, 92:227–239, 2015.

[36] Vasileios Kotronis, Rowan Klöti, Matthias Rost, Panagiotis Georgopoulos, Bernhard Ager, Stefan Schmid, and Xenofontas Dimitropoulos. Stitching inter-domain paths over ixps. In *Proceedings of the Symposium on SDN Research*, pages 1–12, 2016.

[37] Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. *ACM SIGCOMM Computer Communication Review*, 42(4):473–478, 2012.

[38] Paulo Fonseca, Ricardo Bennesby, Edjard Mota, and Alexandre Passito. A replication component for resilient openflow-based networking. In *2012 IEEE Network operations and management symposium*, pages 933–939. IEEE, 2012.

[39] David Hock, Matthias Hartmann, Steffen Gebert, Michael Jarschel, Thomas Zinner, and Phuoc Tran-Gia. Pareto-optimal resilient controller placement in sdn-based core networks. In *Proceedings of the 2013 25th International Teletraffic Congress (ITC)*, pages 1–9. IEEE, 2013.

[40] Cristian Cleder Machado, Lisandro Zambenedetti Granville, and Alberto Schaeffer-Filho. Answer: Combining nfv and sdn features for network resilience strategies. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 391–396. IEEE, 2016.

[41] Xu Zhang, Kefeng Wei, Lei Guo, Weigang Hou, and Jingjing Wu. Sdn-based resilience solutions for smart grids. In *2016 International Conference on Software Networking (ICSN)*, pages 1–5. IEEE, 2016.

[42] MZA Rahman, N Yaakob, A Amir, RB Ahmad, SK Yoon, and AH Abd Halim. Performance analysis of congestion control mechanism in software defined network (sdn). In *MATEC Web of Conferences*, volume 140, page 01033. EDP Sciences, 2017.

[43] Paul Smith, Alberto Schaeffer-Filho, David Hutchison, and Andreas Mauthe. Management patterns: Sdn-enabled network resilience management. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–9. IEEE, 2014.

[44] Production quality, multilayer open virtual switch. URL http://www.openvswitch.org/. Accessed: 2022-10-10.

[45] Kdd cup'99. URL https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[46] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. IEEE, 2009.

[47] The ctu-13 dataset. a labeled dataset with botnet, normal and background traffic. URL https://www.stratosphereips.org/datasets-ctu13.

[48] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6, 2015.

[49] Jiaqi Li, Zhifeng Zhao, and Rongpeng Li. A machine learning based intrusion detection system for software defined 5g network. *arXiv preprint arXiv:1708.04571*, 2017.

[50] Reeta Devi, Rakesh Kumar Jha, Akhil Gupta, Sanjeev Jain, and Preetam Kumar. Implementation of intrusion detection system using adaptive neuro-fuzzy inference system for 5g wireless communication network. *AEU-International Journal of Electronics and Communications*, 74:94–106, 2017.

[51] Hichem Sedjelmaci, Sidi Mohammed Senouci, Nirwan Ansari, and Abdelwahab Boualouache. A trusted hybrid learning approach to secure edge computing. *IEEE Consumer Electronics Magazine*, 2021.

[52] Ibrahim Alrashdi, Ali Alqazzaz, Esam Aloufi, Raed Alharthi, Mohamed Zohdy, and Hua Ming. Ad-iot: Anomaly detection of iot cyberattacks in smart city using machine learning. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0305–0310. IEEE, 2019.

[53] Mahmoud Said Elsayed, Nhien-An Le-Khac, and Anca D Jurcut. Insdn: A novel sdn intrusion dataset. *IEEE Access*, 8:165263–165284, 2020.

[54] Mohammad Rifat Ahmmad Rashid, Giuseppe Rizzo, Marco Torchiano, Nandana Mihindukulasooriya, Oscar Corcho, and Raúl García-Castro. Completeness and consistency analysis for evolving knowledge bases. *Journal of Web Semantics*, 54: 48–71, 2019.

[55] Ahmad Assaf, Raphaël Troncy, and Aline Senart. Roomba: An extensible framework to validate and build dataset profiles. In *The Semantic Web: ESWC 2015 Satellite Events: ESWC 2015 Satellite Events, Portorož, Slovenia, May 31–June 4, 2015, Revised Selected Papers 12*, pages 325–339. Springer, 2015.

[56] Christian Fürber and Martin Hepp. Swiqa–a semantic web information quality assessment framework. 2011.

[57] ibm. What is data quality? | IBM — ibm.com. https://www.ibm.com/topics/data-quality#:~:text=Data%20quality%20is%20a%20broader,accuracy%2C%20consistency%2C%20and%20completeness. [Accessed 10-10-2023].

[58] Jong-Hyouk Lee and Hyoungshick Kim. Security and privacy challenges in the internet of things [security and privacy matters]. *IEEE Consumer Electronics Magazine*, 6(3): 134–136, 2017.

[59] Hongji Huang, Song Guo, Guan Gui, Zhen Yang, Jianhua Zhang, Hikmet Sari, and Fumiyuki Adachi. Deep learning for physical-layer 5g wireless techniques: Opportunities, challenges and solutions. *IEEE Wireless Communications*, 27(1): 214–222, 2019.

[60] Mohamed Amine Ferrag, Leandros Maglaras, Sotiris Moschoyiannis, and Helge Janicke. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *Journal of Information Security and Applications*, 50:102419, 2020.

[61] Alicia Esquivel Morel, Deniz Kavzak Ufuktepe, Robert Ignatowicz, Alexander Riddle, Chengyi Qu, Prasad Calyam, and Kannappan Palaniappan. Enhancing network-edge connectivity and computation security in drone video analytics. In *2020 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, pages 1–12. IEEE, 2020.

[62] Alper Kaan Sarica and Pelin Angin. Explainable security in sdn-based iot networks. *Sensors*, 20(24):7326, 2020.

[63] Ihsan H Abdulqadder, Shijie Zhou, Deqing Zou, Israa T Aziz, and Syed Muhammad Abrar Akber. Multi-layered intrusion detection and prevention in the sdn/nfv enabled cloud of 5g networks using ai-based defense mechanisms. *Computer Networks*, 179:107364, 2020.

[64] He Fang, Xianbin Wang, and Stefano Tomasin. Machine learning for intelligent authentication in 5g and beyond wireless networks. *IEEE Wireless Communications*, 26(5):55–61, 2019.

[65] Lorenzo Fernández Maimó, Félix J García Clemente, Manuel Gil Pérez, and Gregorio Martínez Pérez. On the performance of a deep learning-based anomaly detection system for 5g networks. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1–8. IEEE, 2017.

[66] Lorenzo Fernández Maimó, Alberto Huertas Celdrán, Manuel Gil Pérez, Félix J García Clemente, and Gregorio Martínez Pérez. Dynamic management of a deep learning-based anomaly detection system for 5g networks. *Journal of Ambient Intelligence and Humanized Computing*, 10(8):3083–3097, 2019.

[67] Ji Li, Hui Gao, Tiejun Lv, and Yueming Lu. Deep reinforcement learning based computation offloading and resource allocation for mec. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2018.

[68] Shahadate Rezvy, Yuan Luo, Miltos Petridis, Aboubaker Lasebae, and Tahmina Zebin. An efficient deep learning model for intrusion classification and prediction in 5g and iot networks. In *2019 53rd Annual Conference on information sciences and systems (CISS)*, pages 1–6. IEEE, 2019.

[69] Marouane Hachimi, Georges Kaddoum, Ghyslain Gagnon, and Poulmanogo Illy. Multi-stage jamming attacks detection using deep learning combined with kernelized support vector machine in 5g cloud radio access networks. In *2020 international symposium on networks, computers and communications (ISNCC)*, pages 1–5. IEEE, 2020.

[70] Miquel Puig Mena, Apostolos Papageorgiou, Leonardo Ochoa-Aday, Shuaib Siddiqui, and Gabriele Baldoni. Enhancing the performance of 5g slicing operations via multi-tier orchestration. In *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 131–138. IEEE, 2020.

[71] Adlen Ksentini and Pantelis A Frangoudis. Toward slicing-enabled multi-access edge computing in 5g. *IEEE Network*, 34(2):99–105, 2020.

[72] Luca Cominardi, Thomas Deiss, Miltiadis Filippou, Vincenzo Sciancalepore, Fabio Giust, and Dario Sabella. Mec support for network slicing: Status and limitations from a standardization viewpoint. *IEEE Communications Standards Magazine*, 4(2): 22–30, 2020.

[73] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 183–196, 2017.

[74] Shiao-Li Charles Tsao. Enhanced gtp: an efficient packet tunneling protocol for general packet radio service. In *ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No. 01CH37240)*, volume 9, pages 2819–2823. IEEE, 2001.

[75] Randall Stewart and Christopher Metz. Sctp: new transport protocol for tcp/ip. *IEEE Internet Computing*, 5(6):64–69, 2001.

[76] Pawani Porambage, Tanesh Kumar, Madhusanka Liyanage, Juha Partala, Lauri Lovén, Mika Ylianttila, and Tapio Seppänen. Sec-edgeai: Ai for edge security vs security for edge ai. *The 1st 6G Wireless Summit,(Levi, Finland)*, 2019.

[77] Songlin Chen, Hong Wen, Jinsong Wu, Jie Chen, Wenjie Liu, Lin Hu, and Yi Chen. Physical-layer channel authentication for 5g via machine learning algorithm. *Wireless Communications and Mobile Computing*, 2018, 2018.

[78] Gregory Blanc, Nizar Kheir, Dhouha Ayed, Vincent Lefebvre, Edgardo Montes de Oca, and Pascal Bisson. Towards a 5g security architecture: Articulating software-defined security and security as a service. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 1–8, 2018.

[79] Anastasios Zafeiropoulos, Anastasius Gavras, Anna Tzanakaki, Antonino Albanese, Apostolos Kousaridas, Avi Weit, Bessem Sayadi, Boris Tiomela Jou, Carlos J. Bernardos, Chafika Benzaid, Christian Mannweiler, Daniel Camps-Mur, David Breitgand, David Gutierrez Estevez, David Navratil, De Mi, Diego Lopez, Dimitrios Klonidis, Edward Mutafungwa, Eleni Fotopoulou, Emmanouil Kafetzakis, Emmanouil Pateromichelakis, Erez Biton, Fasil B. Tesema, George Kalfas, Holger Karl, Jens Bartelt, Jesús Gutiérrez, John Cosmas, John Thomson, Jordi Joan Giménez, Jose M. Alcaraz Calero, Josep Mangues-Bafalluy, Kostas Katsalis, Laurent Gallo, Marco Gramaglia, Maria Rita Spada, Mukhald Salih, Navid Nikaein, Nawar Jawad, Nebojsa Maletic, Ömer Bulakci, Panagiotis Demestichas, Peer Hasselmeyer, Qi Wang, Qing Wei, Refik Fatih Ustok, Rolf Blom, Salvatore Pontarelli, Selçuk Keskin, Stefano Salsano, Stephanie Parker, Thomas Deiss, Ugur Acar, Xi Li, and Yue Zhang. *5G PPP Architecture Working Group: View on 5G Architecture*, volume Version 3.0. European Commission, Belgium, June 2019.

[80] Keke Gai, Meikang Qiu, Lixin Tao, and Yongxin Zhu. Intrusion detection techniques for mobile cloud computing in heterogeneous 5g. *Security and communication networks*, 9(16):3049–3058, 2016.

[81] Omec, Aug 2020. URL https://opennetworking.org/omec/.

[82] Ns3-5g-lena. URL https://apps.nsnam.org/app/nr/.

[83] Admin and Andras. Omnet. URL https://omnetpp.org/.

[84] Mininet. Mininet. URL http://mininet.org/.

[85] Muhammad Imran, Abas Md Said, and Halabi Hasbullah. A survey of simulators, emulators and testbeds for wireless sensor networks. In *2010 International Symposium on Information Technology*, volume 2, pages 897–902. IEEE, 2010.

[86] 5g core network. URL https://openairinterface.org/oai-5g-core-network-project/.

[87] Navid Nikaein, Raymond Knopp, Florian Kaltenberger, Lionel Gauthier, Christian Bonnet, Dominique Nussbaum, and Riadh Ghaddab. Openairinterface: an open lte network in a pc. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 305–308, 2014.

[88] Florian Kaltenberger, Aloizio P Silva, Abhimanyu Gosain, Luhan Wang, and Tien-Thinh Nguyen. Openairinterface: Democratizing innovation in the 5g era. *Computer Networks*, 176:107284, 2020.

[89] Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang, Johari Abdullah, and Farhan Ahmad. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32(1):e4150, 2021.

[90] Philokypros Ioulianou, Vasileios Vasilakis, Ioannis Moscholios, and Michael Logothetis. A signature-based intrusion detection system for the internet of things. *Information and Communication Technology Form*, 2018.

[91] Marcus Botacin, Marco Zanata Alves, Daniela Oliveira, and André Grégio. Heaven: A hardware-enhanced antivirus engine to accelerate real-time, signature-based malware detection. *Expert Systems with Applications*, 201:117083, 2022.

[92] Dongzi Jin, Yiqin Lu, Jiancheng Qin, Zhe Cheng, and Zhongshu Mao. Swiftids: Real-time intrusion detection system based on lightgbm and parallel intrusion detection mechanism. *Computers & Security*, 97:101984, 2020.

[93] Kuruva Lakshmanna, Rajesh Kaluri, Nagaraja Gundluru, Zamil S Alzamil, Dharmendra Singh Rajput, Arfat Ahmad Khan, Mohd Anul Haq, and Ahmed Alhussen. A review on deep learning techniques for iot data. *Electronics*, 11(10):1604, 2022.

[94] Ali Imran, Ahmed Zoha, and Adnan Abu-Dayya. Challenges in 5g: how to empower son with big data for enabling 5g. *IEEE network*, 28(6):27–33, 2014.

[95] Asmaa Halbouni, Teddy Surya Gunawan, Mohamed Hadi Habaebi, Murad Halbouni, Mira Kartiwi, and Robiah Ahmad. Machine learning and deep learning approaches for cybersecuriy: A review. *IEEE Access*, 2022.

[96] Thenmozhi Rayan and SC Sandeep. Machine learning ids models for 5g and iot. *Secure Communication for 5G and IoT Networks*, pages 73–84, 2022.

[97] Dynamic Designz. Dynamicdesignz/hoic: High orbit ion cannon. URL https://github.com/DynamicDesignz/HOIC.

[98] NewEra Cracker. Neweracracker/loic: Low orbit ion cannon - an open source network stress tool, written in c. based on praetox's loic project. use on your own risk. without any express or implied warranties. URL https://github.com/NewEraCracker/LOIC.

[99] hping3. URL https://tools.kali.org/information-gathering/hping3.

[100] Cheng Jin, Haining Wang, and Kang G Shin. Hop-count filtering: an effective defense against spoofed ddos traffic. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 30–41, 2003.

[101] Yuchuan Deng, Hao Jiang, Peijing Cai, Tong Wu, Pan Zhou, Beibei Li, Hao Lu, Jing Wu, Xin Chen, and Kehao Wang. Resource provisioning for mitigating edge ddos attacks in mec-enabled sdvn. *IEEE Internet of Things Journal*, 9(23):24264–24280, 2022.

[102] Pasika Ranaweera, Anca Jurcut, and Madhusanka Liyanage. Mec-enabled 5g use cases: a survey on security vulnerabilities and countermeasures. *ACM Computing Surveys (CSUR)*, 54(9):1–37, 2021.

[103] Ijaz Ahmad, Sergio Lembo, Felipe Rodriguez, Stephan Mehnert, and Mikko Vehkaperä. Security of micro mec in 6g: A brief overview. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, pages 332–337. IEEE, 2022.

[104] Shin-Ming Cheng, Bing-Kai Hong, and Cheng-Feng Hung. Attack detection and mitigation in mec-enabled 5g networks for aiot. *IEEE Internet of Things Magazine*, 5 (3):76–81, 2022.

[105] Arpit Tripathi, Abhishek Thakur, and Bheemarjuna Reddy Tamma. Attack graphs for standalone non-public 5g networks. In *2022 IEEE Future Networks World Forum (FNWF)*, pages 158–163. IEEE, 2022.

[106] Sergio Ruiz-Villafranca, José Roldán-Gómez, Javier Carrillo-Mondéjar, Juan Manuel Castelo Gómez, and José Miguel Villalón. A mec-iiot intelligent threat detector based on machine learning boosted tree algorithms. *Computer Networks*, page 109868, 2023.

[107] Mojtaba Eskandari, Zaffar Haider Janjua, Massimo Vecchio, and Fabio Antonelli. Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices. *IEEE Internet of Things Journal*, 7(8):6882–6897, 2020.

[108] Vasaka Visoottiviseth, Pranpariya Sakarin, Jetnipat Thongwilai, and Thanakrit Choobanjong. Signature-based and behavior-based attack detection with machine learning for home iot devices. In *2020 IEEE REGION 10 CONFERENCE (TENCON)*, pages 829–834. IEEE, 2020.

[109] Jesús Díaz-Verdejo, Javier Muñoz-Calle, Antonio Estepa Alonso, Rafael Estepa Alonso, and Germán Madinabeitia. On the detection capabilities of signature-based intrusion detection systems in the context of web attacks. *Applied Sciences*, 12 (2):852, 2022.

[110] Mansoor Farooq. Supervised learning techniques for intrusion detection system based on multi-layer classification approach. *International Journal of Advanced Computer Science and Applications*, 13(3), 2022.

[111] T Yerriswamy and Gururaj Murtugudde. Signature-based traffic classification for ddos attack detection and analysis of mitigation for ddos attacks using programmable commodity switches. *International Journal of Performability Engineering*, 18(7):529, 2022.

[112] Abdul Waleed, Abdul Fareed Jamali, and Ammar Masood. Which open-source ids? snort, suricata or zeek. *Computer Networks*, 213:109116, 2022.

[113] Ye-Eun Kim, Yea-Sul Kim, and Hwankuk Kim. Effective feature selection methods to detect iot ddos attack in 5g core network. *Sensors*, 22(10):3819, 2022.

[114] Rongpeng Li, Zhifeng Zhao, Xuan Zhou, Guoru Ding, Yan Chen, Zhongyao Wang, and Honggang Zhang. Intelligent 5g: When cellular networks meet artificial intelligence. *IEEE Wireless communications*, 24(5):175–183, 2017.

[115] Lorenzo Fernandez Maimo, Alberto Huertas Celdran, Angel L Perales Gomez, Felix J Garcia Clemente, James Weimer, and Insup Lee. Intelligent and dynamic ransomware spread detection and mitigation in integrated clinical environments. *Sensors*, 19(5): 1114, 2019.

[116] Syed Ali Raza Shah and Biju Issac. Performance comparison of intrusion detection systems and application of machine learning to snort system. *Future Generation Computer Systems*, 80:157–170, 2018.

[117] Nitasha Sahani, Ruoxi Zhu, Jin-Hee Cho, and Chen-Ching Liu. Machine learning-based intrusion detection for smart grid computing: A survey. *ACM Transactions on Cyber-Physical Systems*, 2023.

[118] Hichem Sedjelmaci. Cooperative attacks detection based on artificial intelligence system for 5g networks. *Computers & Electrical Engineering*, 91:107045, 2021.

[119] Shun-Sheng Wang, Kuo-Qin Yan, Shu-Ching Wang, and Chia-Wei Liu. An integrated intrusion detection system for cluster-based wireless sensor networks. *Expert Systems with Applications*, 38(12):15234–15243, 2011.

[120] James B Fraley and James Cannady. The promise of machine learning in cybersecurity. In *SoutheastCon 2017*, pages 1–6. IEEE, 2017.

[121] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. *Ieee access*, 6:35365–35381, 2018.

[122] Roberto Magán-Carrión, Daniel Urda, Ignacio Díaz-Cano, and Bernabé Dorronsoro. Towards a reliable comparison and evaluation of network intrusion detection systems based on machine learning approaches. *Applied Sciences*, 10(5):1775, 2020.

[123] T Saranya, S Sridevi, C Deisy, Tran Duc Chung, and MKA Ahamed Khan. Performance analysis of machine learning algorithms in intrusion detection system: A review. *Procedia Computer Science*, 171:1251–1260, 2020.

[124] Jan Lansky, Saqib Ali, Mokhtar Mohammadi, Mohammed Kamal Majeed, Sarkhel H Taher Karim, Shima Rashidi, Mehdi Hosseinzadeh, and Amir Masoud Rahmani. Deep learning-based intrusion detection systems: a systematic review. *IEEE Access*, 9:101574–101599, 2021.

[125] Hyun Min Song, Jiyoung Woo, and Huy Kang Kim. In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications*, 21: 100198, 2020.

[126] B Riyaz and Sannasi Ganapathy. A deep learning approach for effective intrusion detection in wireless networks using cnn. *Soft Computing*, 24(22):17265–17278, 2020.

[127] Muder Almiani, Alia AbuGhazleh, Amer Al-Rahayfeh, Saleh Atiewi, and Abdul Razaque. Deep recurrent neural network for iot intrusion detection system. *Simulation Modelling Practice and Theory*, 101:102031, 2020.

[128] Shifu Hou, Aaron Saas, Lifei Chen, and Yanfang Ye. Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs. In *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)*, pages 104–111. IEEE, 2016.

[129] Shaveta Dargan, Munish Kumar, Maruthi Rohit Ayyagari, and Gulshan Kumar. A survey of deep learning and its applications: a new paradigm to machine learning. *Archives of Computational Methods in Engineering*, 27(4):1071–1092, 2020.

[130] Arwa Aldweesh, Abdelouahid Derhab, and Ahmed Z Emam. Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. *Knowledge-Based Systems*, 189:105124, 2020.

[131] William Grant Hatcher and Wei Yu. A survey of deep learning: Platforms, applications and emerging research trends. *IEEE Access*, 6:24411–24432, 2018.

[132] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

[133] Amjad Rehman Khan, Muhammad Kashif, Rutvij H Jhaveri, Roshani Raut, Tanzila Saba, and Saeed Ali Bahaj. Deep learning for intrusion detection and security of internet of things (iot): current analysis, challenges, and possible solutions. *Security and Communication Networks*, 2022, 2022.

[134] Yalei Ding and Yuqing Zhai. Intrusion detection system for nsl-kdd dataset using convolutional neural networks. In *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*, pages 81–85, 2018.

[135] Ishan Prakash, Aniruddh Bansal, Rohit Verma, and Rajeev Shorey. Smartsplit: Latency-energy-memory optimisation for cnn splitting on smartphone environment. In *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 549–557. IEEE, 2022.

[136] Hasib-Al Rashid, Utteja Kallakuri, and Tinoosh Mohsenin. Tinym2net-v2: A compact low power software hardware architecture for m ulti m odal deep neural networks. *ACM Transactions on Embedded Computing Systems*, 2023.

[137] Xing Xu, Jie Li, Yang Yang, and Fumin Shen. Toward effective intrusion detection using log-cosh conditional variational autoencoder. *IEEE Internet of Things Journal*, 8(8):6187–6196, 2020.

[138] Meliboev Azizjon, Alikhanov Jumabek, and Wooseong Kim. 1d cnn based network intrusion detection with normalization on imbalanced data. In *2020 international conference on artificial intelligence in information and communication (ICAIIC)*, pages 218–224. IEEE, 2020.

[139] Leila Mohammadpour, Teck Chaw Ling, Chee Sun Liew, and Alihossein Aryanfar. A survey of cnn-based network intrusion detection. *Applied Sciences*, 12(16):8162, 2022.

[140] Jiyeon Kim, Jiwon Kim, Hyunjung Kim, Minsun Shim, and Eunjung Choi. Cnn-based network intrusion detection against denial-of-service attacks. *Electronics*, 9(6):916, 2020.

[141] Suwani Jayasinghe, Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, and Mika Ylianttila. Federated learning based anomaly detection as an enabler for securing network and service management automation in beyond 5g networks. In *2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 345–350. IEEE, 2022.

[142] Muhammad Ahmad, Qaiser Riaz, Muhammad Zeeshan, Hasan Tahir, Syed Ali Haider, and Muhammad Safeer Khan. Intrusion detection in internet of things using supervised machine learning based on application and transport layer features using unsw-nb15 data-set. *EURASIP Journal on Wireless Communications and Networking*, 2021(1): 1–23, 2021.

[143] Muhammad Zeeshan, Qaiser Riaz, Muhammad Ahmad Bilal, Muhammad K Shahzad, Hajira Jabeen, Syed Ali Haider, and Azizur Rahim. Protocol-based deep intrusion detection for dos and ddos attacks using unsw-nb15 and bot-iot data-sets. *IEEE Access*, 10:2269–2283, 2021.

[144] Ingyom Kim and Tai-Myoung Chung. Malicious-traffic classification using deep learning with packet bytes and arrival time. In *International Conference on Future Data and Security Engineering*, pages 345–356. Springer, 2020.

[145] Chuan Yue, Lide Wang, Dengrui Wang, Ruifeng Duo, and Xiaobo Nie. An ensemble intrusion detection method for train ethernet consist network based on cnn and rnn. *IEEE Access*, 9:59527–59539, 2021.

[146] Yang Liu, Jian Kang, Yiran Li, and Bin Ji. A network intrusion detection method based on cnn and cbam. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2021.

[147] Tal Shapira and Yuval Shavitt. Flowpic: Encrypted internet traffic classification is as easy as image recognition. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 680–687. IEEE, 2019.

[148] Sydney Mambwe Kasongo and Yanxia Sun. A deep learning method with wrapper based feature extraction for wireless intrusion detection system. *Computers & Security*, 92:101752, 2020.

[149] Hyun-Jin Kim, Jonghoon Lee, Cheolhee Park, and Jong-Geun Park. Network anomaly detection based on domain adaptation for 5g network security. In *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, pages 976–980. IEEE, 2022.

[150] Shanshuo Ding, Liang Kou, and Ting Wu. A gan-based intrusion detection model for 5g enabled future metaverse. *Mobile Networks and Applications*, 27(6):2596–2610, 2022.

[151] Lifeng Lei, Liang Kou, Xianghao Zhan, Jilin Zhang, and Yongjian Ren. An anomaly detection algorithm based on ensemble learning for 5g environment. *Sensors*, 22(19): 7436, 2022.

[152] Omesh A Fernando, Hannan Xiao, and Joseph Spring. Developing a testbed with p4 to generate datasets for the analysis of 5g-mec security. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 2256–2261. IEEE, 2022.

[153] Kehe Wu, Zuge Chen, and Wei Li. A novel intrusion detection model for a massive network using convolutional neural networks. *Ieee Access*, 6:50850–50859, 2018.

[154] Yihan Xiao, Cheng Xing, Taining Zhang, and Zhongkai Zhao. An intrusion detection model based on feature reduction and convolutional neural networks. *IEEE Access*, 7: 42210–42219, 2019.

[155] Li Yong and Zhang Bo. An intrusion detection model based on multi-scale cnn. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 214–218. IEEE, 2019.

[156] R Vinayakumar, KP Soman, and Prabaharan Poornachandran. Applying convolutional neural network for network intrusion detection. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1222–1228. IEEE, 2017.

[157] Jianjing Cui, Jun Long, Erxue Min, Qiang Liu, and Qian Li. Comparative study of cnn and rnn for deep learning based intrusion detection system. In *International Conference on Cloud Computing and Security*, pages 159–170. Springer, 2018.

[158] Minh Doan and Zhanyang Zhang. Deep learning in 5g wireless networks-anomaly detections. In *2020 29th Wireless and Optical Communications Conference (WOCC)*, pages 1–6. IEEE, 2020.

[159] Davide Sanvito, Daniele Moro, Mattia Gulli, Ilario Filippini, Antonio Capone, and Andrea Campanella. Onos intent monitor and reroute service: enabling plug&play routing logic. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 272–276. IEEE, 2018.

[160] onos. Onos project. URL https://wiki.onosproject.org/.

[161] Lucas V Ruchel, Rogério C Turchetti, and Edson T de Camargo. Evaluation of the robustness of sdn controllers onos and odl. *Computer Networks*, 219:109403, 2022.

[162] Lusani Mamushiane and Themba Shozi. A qos-based evaluation of sdn controllers: Onos and opendaylight. In *2021 IST-Africa Conference (IST-Africa)*, pages 1–10. IEEE, 2021.

[163] P4 Language Consortium et al. Behavioral model (bmv2). *URL: https://github.com/p4lang/behavioral-model [cited 2020-01-21]*, 2018.

[164] Mohammad Alizadeh and Tom Edsall. On the data path performance of leaf-spine datacenter fabrics. In *2013 IEEE 21st annual symposium on high-performance interconnects*, pages 71–74. IEEE, 2013.

[165] OA Fernando, Hannan Xiao, and Xianhui Che. Evaluation of underlying switching mechanism for future networks with p4 and sdn (workshop paper). In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 549–568. Springer, 2019.

[166] Saba Al-Rubaye, Ekhlas Kadhum, Qiang Ni, and Alagan Anpalagan. Industrial internet of things driven by sdn platform for smart grid resiliency. *IEEE Internet of Things Journal*, 6(1):267–277, 2017.

[167] Nteziriza Nkerabahizi Josbert, Wang Ping, Min Wei, and Yong Li. Industrial networks driven by sdn technology for dynamic fast resilience. *Information*, 12(10):420, 2021.

[168] Ellen W Zegura, Kenneth L Calvert, and Michael J Donahoo. A quantitative comparison of graph-based models for internet topology. *IEEE/ACM Transactions on networking*, 5(6):770–783, 1997.

[169] Romualdo Pastor-Satorras and Alessandro Vespignani. *Evolution and structure of the Internet: A statistical physics approach.* Cambridge University Press, 2007.

[170] Faria Khandaker, Sharief Oteafy, Hossam S Hassanein, and Hesham Farahat. A functional taxonomy of caching schemes: Towards guided designs in information-centric networks. *Computer Networks*, 165:106937, 2019.

[171] Binxu Yang, Wei Koong Chai, Zichuan Xu, Konstantinos V Katsaros, and George Pavlou. Cost-efficient nfv-enabled mobile edge-cloud for low latency mobile applications. *IEEE Transactions on Network and Service Management*, 15(1):475–488, 2018.

[172] Fernando Matos, Alexandre Matos, Paulo Simoes, and Edmundo Monteiro. Provisioning of inter-domain qos-aware services. *Journal of Computer Science and Technology*, 30(2):404–420, 2015.

[173] Stênio Fernandes. Methods and techniques for measurements in the internet. In *Performance Evaluation for Network Services, Systems and Protocols*, pages 45–73. Springer, 2017.

[174] Guanrong Chen, Zhengping Fan, and Xiang Li. Modelling the complex internet topology. In *Complex Dynamics in Communication Networks*, pages 213–234. Springer, 2005.

[175] Jon Postel et al. Transmission control protocol. 1981.

[176] Jon Postel. Rfc0768: User datagram protocol, 1980.

[177] Esma Yildirim, Ibrahim H Suslu, and Tevfik Kosar. Which network measurement tool is right for you? a multidimensional comparison study. In *2008 9th IEEE/ACM International Conference on Grid Computing*, pages 266–275. IEEE, 2008.

[178] Eren Balevi and Richard D Gitlin. Unsupervised machine learning in 5g networks for low latency communications. In *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, pages 1–2. IEEE, 2017.

[179] Stefan Saroiu, Krishna P Gummadi, Richard J Dunn, Steven D Gribble, and Henry M Levy. An analysis of internet content delivery systems. *ACM SIGOPS Operating Systems Review*, 36(SI):315–327, 2002.

[180] Vivien GUEANT. iperf - the ultimate speed test tool for tcp, udp and sctptest the limits of your network + internet neutrality test. URL https://iperf.fr/.

[181] VideoLAN. Vlc media player for ubuntu. URL https://www.videolan.org/vlc/download-ubuntu.html. Accessed: 2020-04-10.

[182] Roberto Bifulco, Julien Boite, Mathieu Bouet, and Fabian Schneider. Improving sdn with inspired switches. In *Proceedings of the Symposium on SDN Research*, pages 1–12, 2016.

[183] Aliyu Lawal Aliyu, Peter Bull, and Ali Abdallah. Performance implication and analysis of the openflow sdn protocol. In *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 391–396. IEEE, 2017.

[184] Gianni Antichi, Ignacio Castro, Marco Chiesa, Eder L Fernandes, Remy Lapeyrade, Daniel Kopp, Jong Hun Han, Marc Bruyere, Christoph Dietzel, Mitchell Gusat, et al. Endeavour: A scalable sdn architecture for real-world ixps. *IEEE Journal on Selected Areas in Communications*, 35(11):2553–2562, 2017.

[185] Vasileios Giotsas, Christoph Dietzel, Georgios Smaragdakis, Anja Feldmann, Arthur Berger, and Emile Aben. Detecting peering infrastructure outages in the wild. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 446–459, 2017.

[186] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. Enabling fast failure recovery in openflow networks. In *2011 8th International Workshop on the Design of Reliable Communication Networks (DRCN)*, pages 164–171. IEEE, 2011.

[187] Olatunde Awobuluyi. Periodic control update overheads in openflow-based enterprise networks. In *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, pages 390–396. IEEE, 2014.

[188] Matej Groma, Tomáš Boros, and Pavol Helebrandt. Scalable cache-based address resolution protocol handling in software-defined networks. In *2019 XXVII International Conference on Information, Communication and Automation Technologies (ICAT)*, pages 1–6. IEEE, 2019.

[189] Yuan-Cheng Lai, Ahsan Ali, Md Shohrab Hossain, and Ying-Dar Lin. Performance modeling and analysis of tcp and udp flows over software defined networks. *Journal of Network and Computer Applications*, 130:76–88, 2019.

[190] Francesco Paolucci, Filippo Cugini, Piero Castoldi, and Tomasz Osiński. Enhancing 5g sdn/nfv edge with p4 data plane programmability. *IEEE Network*, 35(3):154–160, 2021.

[191] Opencells-programmable sim cards. URL https://open-cells.com/.

[192] oai / openairinterface5g. URL https://gitlab.eurecom.fr/oai/openairinterface5g.

[193] Usrp sdr x310. URL https://files.ettus.com/manual/page_usrp_x3x0.html.

[194] Virtualization - uvt. URL https://ubuntu.com/server/docs/virtualization-uvt.

[195] Peng Liu, Bozhao Qi, and Suman Banerjee. Edgeeye: An edge service framework for real-time intelligent video analytics. In *Proceedings of the 1st international workshop on edge systems, analytics and networking*, pages 1–6, 2018.

[196] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. Videoedge: Processing camera streams using hierarchical clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 115–131. IEEE, 2018.

[197] Angel Martin, Roberto Viola, Mikel Zorrilla, Julián Flórez, Pablo Angueira, and Jon Montalbán. Mec for fair, reliable and efficient media streaming in mobile networks. *IEEE Transactions on Broadcasting*, 66(2):264–278, 2019.

[198] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[199] Robert Sedgewick and Kevin Wayne. *Algorithms: Part I*. Addison-Wesley Professional, 2014.

[200] Micha Gorelick and Ian Ozsvald. *High Performance Python: Practical Performant Programming for Humans*. O'Reilly Media, 2020.

[201] Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[202] Shivam Sinha, TN Singh, VK Singh, and AK Verma. Epoch determination for neural network by self-organized map (som). *Computational Geosciences*, 14:199–206, 2010.

[203] Dimitrios Papamartzivanos, Félix Gómez Mármol, and Georgios Kambourakis. Dendron: Genetic trees driven rule induction for network intrusion detection systems. *Future Generation Computer Systems*, 79:558–574, 2018.

[204] Ravi Vinayakumar, Mamoun Alazab, KP Soman, Prabaharan Poornachandran, Ameer Al-Nemrat, and Sitalakshmi Venkatraman. Deep learning approach for intelligent intrusion detection system. *IEEE Access*, 7:41525–41550, 2019.

[205] Nour Moustafa and Jill Slay. The significant features of the unsw-nb15 and the kdd99 data sets for network intrusion detection systems. In *2015 4th International workshop on building analysis datasets and gathering experience returns for security (BADGERS)*, pages 25–31. IEEE, 2015.

[206] Chaouki Khammassi and Saoussen Krichen. A ga-lr wrapper approach for feature selection in network intrusion detection. *computers & security*, 70:255–277, 2017.

[207] Haotian Zhang, Lin Zhang, and Yuan Jiang. Overfitting and underfitting analysis for deep learning based end-to-end communication systems. In *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6. IEEE, 2019.

[208] Tomasz W Nowak, Mariusz Sepczuk, Zbigniew Kotulski, Wojciech Niewolski, Rafal Artych, Krzysztof Bocianiak, Tomasz Osko, and Jean-Philippe Wary. Verticals in 5g mec-use cases and security challenges. *IEEE Access*, 9:87251–87298, 2021.

[209] Pedro Cruz, Nadjib Achir, and Aline Carneiro Viana. On the edge of the deployment: A survey on multi-access edge computing. *ACM Computing Surveys*, 55(5):1–34, 2022.

[210] Scott A Brandt, Scott Banachowski, Caixue Lin, and Timothy Bisson. Dynamic integrated scheduling of hard real-time, soft real-time, and non-real-time processes. In *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, pages 396–407. IEEE, 2003.

[211] Pyshark. URL https://pypi.org/project/pyshark/.

[212] Saikat Guha, Yutaka Takeda, and Paul Francis. Nutss: A sip-based approach to udp and tcp network connectivity. In *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 43–48, 2004.

[213] Fuse filesystem over google drive. URL https://github.com/astrada/google-drive-ocamlfuse. Accessed: 2022-10-28.

[214] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks*, 51(12):3448–3470, 2007.

[215] Omesh A Fernando, Hannan Xiao, and Joseph Spring. New algorithms for the detection of malicious traffic in 5g-mec. In *2023 IEEE Wireless Communications and Networking Conference (WCNC), 26–29 March 2023, Glasgow, Scotland, UK*. IEEE, 2023.

[216] Konstantinos Poularakis, Jaime Llorca, Antonia M Tulino, Ian Taylor, and Leandros Tassiulas. Service placement and request routing in mec networks with storage, computation, and communication constraints. *IEEE/ACM Transactions on Networking*, 28(3):1047–1060, 2020.

[217] Intel. Explore the power of intel® intelligent fabric processors. URL https://www.intel.co.uk/content/www/uk/en/products/network-io/programmable-ethernet-switch.html.

[218] Xiaoquan Zhang, Lin Cui, Fung Po Tso, and Weijia Jia. Compiling service function chains via fine-grained composition in the programmable data plane. *IEEE Transactions on Services Computing*, 2023.

[219] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.