

https://dspace.org/

Controller-plane workload characterization and forecasting in software-defined networking

Nkosi, Emmanuel

2017

http://hdl.handle.net/10539/22955

Nkosi, Emmanuel (2016) Controller-plane workload characterization and forecasting in software-defined networking, University of the Witwatersrand, Johannesburg, <http://hdl.handle.net/10539/22955>

Downloaded from a DSpace repository. https://dspace.org/

CONTROLLER-PLANE WORKLOAD CHARACTERIZATION AND FORECASTING IN SOFTWARE-DEFINED NETWORKING

Emmanuel Nkosi

296390

A research report submitted to the Faculty of Engineering and the Built Environment of the University of the Witwatersrand in partial fulfilment of the requirements for the degree of

Master of Science in Engineering

February 2017

Contents

Declaration vii

Acknowledgements viii

Abstract ix

List of Figures ix

List of Tables xiii

Acronyms xvi

Symbols xvii

1 Introduction 1

1.1 Transformation of the Telecoms Network 2

1.2 Problems Facing Traditional IP Networks 4

1.2.1 Complexity in Network Configuration and Management 4

1.2.2 Vertical Integration in Networking Devices 5

1.3 The SDN Promise 6

1.3.1 SDN Architecture and Principle 7

1.3.2 Benefits of SDN 9

1.4 State of the Art: Southbound Interface in SDN 11
1.4.1 Data plane 11
1.4.2 Southbound Interface Protocol: OpenFlow 12
1.4.3 Secure OpenFlow Channel 16
1.5 The Research Question 17
1.6 Contribution of the Research 21
1.7 Structure of the Report 23

2 Literature Survey **24**

2.1 SDN Controllers 26
2.1.1 An Operating System for Networks (NOX) 27
2.1.2 Maestro 28
2.1.3 Beacon 29
2.1.4 Floodlight 30
2.1.5 Trema 31
2.1.6 Summary on SDN Controllers 32
2.2 Distributed and Hierarchical Controller Planes 32
2.2.1 A Distributed Control Platform: Onix 32
2.2.2 Open Network Operating System: Onos 34
2.2.3 HyperFlow 35
2.2.4 OpenDayLight 36
2.2.5 Hierarchical Controller planes: Kandoo 37

2.2.6 Summary on Distributed Controller planes 38
2.3 SDN Controller Performance Measurement Tool 39
2.4 Traffic Analysis in SDN 40
2.4.1 NetFlow 41
2.4.2 Planck 41
2.4.3 Payless 42
2.4.4 OpenTM 43
2.4.5 Summary on Traffic Monitoring Tools 43
2.5 SDN Traffic Engineering Schemes And Other Solutions 44
2.5.1 Aster*x 44
2.5.2 SDN Controller Placement 45
2.5.3 Scotch 46
2.5.4 ElastiCon 47
2.6 Chapter Summary 48

3 Network Prototype **50**

3.1 Virtualization in Network Emulators 53
3.2 Container-Based Network Emulator: Mininet 55
3.3 Data Centers 57
3.3.1 Data Center Topology 58
3.3.2 Data Center Traffic Generation 59
3.4 Playing Traffic in Mininet 62

3.4.1 Execution of programs in Mininet Hosts 63
3.4.2 A Multi-threaded Software Tool to Replay Traffic 64
3.5 Chapter Summary 68

4 Characterisation of Controller Workload **69**

4.1 Fitting Statistical Distributions 70
4.1.1 Selection of Candidate Distributions 71
4.1.2 Assessment of Goodness-of-fit 73
4.2 Benchmarking and Comparisons of Results 74
4.3 Description of Observed Data 76
4.4 Distributions of Inter-arrival Times 79
4.4.1 Flow Inter-arrival Times CDFs 80
4.4.2 Fitting Data to Known Probability Distributions 82
4.5 Evaluation of Results 88
4.6 Chapter Summary 90

5 Forecasting of Controller Workload **91**

5.1 Time Series Analysis 92
5.2 Autoregressive Integrated Moving Average Models 94
5.3 Artificial Neural Network 96
5.4 Forecast Model Evaluation and Forecast Accuracy 98
5.5 Benchmarking and Comparisons of Results 102
5.6 Forecasting Traffic Intensity 103

5.6.1 Auto-Regressive Integrated Moving Average Models 107

5.6.2 Artificial Neural Network (ANN) Models 114

5.7 Evaluation of Results 121

5.8 Chapter Summary 125

6 Conclusion **127**

6.1 Research Contribution 127

6.2 Future Work 130

Bibliography **131**

DECLARATION

I hereby declare that the work contained in this research report is my own. It is being submitted to the Faculty of Engineering and the Built Environment of the University of the Witwatersrand in partial fulfilment of the requirements for the degree of Master of Science in Engineering. It has not been submitted for examination in any other University or institution before.

.....
(Signature of Candidate)

on this ..... day of ..... year .....

ACKNOWLEDGMENTS

In His Greatness, the Lord God made His wonderful works to be remembered –Studied by all who have pleasure therein. (Adapted from Psalm 111:2). I dedicate the successful completion of this work to the Lord Jesus Christ, to the glory of God the Father.

I received support and assistance from a number of individuals throughout the programme of this research work.

I would like to thank my supervisor, Professor Ling Cheng for the support and guidance he provided – Prof., thanks for understanding throughout.

Being a part-time student, I work for the Regional Integrated Network Planning (RINP NER) department in Telkom SA. I would like to thank all my colleagues for their support and assistance. More importantly, I would like to thank our Manager Charles du Preez for the time he afforded me to concentrate on my studies while I also had to report to work.

I would also like to thank Malifetsane Mokoena for spending time in proof-reading the research report.

I heard a sound of a heavy rain

I beheld the beauty of the clouds

It poured down for everyone

For everything

Even for me

And he said: "Mine From the LORD"

Ngiyabonga | Kea Leboha | Dankie | Thank you | Ndo Livhuwa

Amen

ABSTRACT

Software-defined networking (SDN) is the physical separation of the control and data planes in networking devices. A logically centralised controller plane which uses a network-wide view data structure to control several data plane devices is another defining attribute of SDN. The centralised controllers and the network-wide view data structure are difficult to scale as the network and the data it carries grow. Solutions which have been proposed to combat this challenge in SDN lack the use of the statistical properties of the workload or network traffic seen by SDN controllers. Hence, the objective of this research is twofold: Firstly, the statistical properties of the controller workload are investigated. Secondly, Autoregressive Integrated Moving Average Models (ARIMA) and Artificial Neural Network (ANN) models are investigated to establish the feasibility of forecasting the controller workload signal. Representations of the state of the controller plane in the network-wide view in the form of forecasts of the controller workload will enable control applications to detect dwindling controller resources and therefore alleviate controller congestion. On the other hand, realistic statistical traffic models of the controller workload variable are sought for the design and evaluation of SDN controllers. A data center network prototype is created by making use of an SDN network emulator called Mininet and an SDN controller called Onos. It was found that 1–2% of flows arrive within 10μs of each other and more than 80% have inter-arrival times in the range of 10μs–10ms. These inter-arrival times were found to follow a beta distribution, which is similar to findings made in Machine Type Communications (MTC). The use of ARIMA and ANN to forecast the controller workload established that it is feasible to forecast the workload seen by SDN controllers. The accuracy of these models was found to be comparable for continuously valued time series signals. The ANN model was found to be applicable even in discretely valued time series data.

List of Figures

1.1 An Illustration of a Network Devices's Functional Planes – Adapted From [1]. 5

1.2 The Three Layered SDN Architecture – Adapted From [2]. 8

1.3 OpenFlow 1.1.0 Switch Components – Adapted From [3]. 11

1.4 OpenFlow Basic Packet Matching Process – Adapted From [4]. 13

1.5 OpenFlow 1.5.1 Switch Components – Adapted From [5]. 14

1.6 Reactive Forwarding in OpenFlow Switches – Adapted From [4]. 17

1.7 OpenFlow Basic Packet Matching Process. 19

2.1 The 4D Architecture Divided to Indicate the 3 Layers of the SDN Concept – Adapted From [6]. 25

3.1 Hypervisor-Based Virtualization – Adapted From [7]. 54

3.2 OS-level Virtualization – Adapted From [7]. 54

3.3 A 3-tier Data Center Network Topology – Adapted From [8]. 58

3.4 A Fat Tree Topology of $K = 4$. 63

3.5 An Illustration of the Logic and Execution of the Traffic Replaying Software. 67

4.1 Example of Empirical Density and Cumulative Plots. 71

4.2 Example of a Cullen and Frey Graph. 73

4.3 A Time Series of `packet in` Messages With Observations Made at 1ms Intervals. 75

4.4 A Time Series of `packet in` Messages With Observations Made at 10ms Intervals. 75

4.5 A Time Series of `packet in` Messages With Observations Made at 1s intervals. 76

4.6 Data Capture Illustration. 77

4.7 Flow Inter-arrival Times CDF. 81

4.8 Skewness and Kurtosis Plot: One Hundred Flows. 83

4.9 Skewness and Kurtosis Plot: One Thousand Flows. 83

4.10 Fitted Gamma and Beta Distributions. 86

5.1 An Illustration of an ANN Model – Adapted From [9]. 97

5.2 A Time Series of `packet in` Messages With Observations Made at 1ms Intervals. 104

5.3 A Time Series of `packet in` Messages With Observations Made at 10ms Intervals. 104

5.4 A Time Series of `packet in` Messages With Observations Made at 1s Intervals. 105

5.5 ACF Plot of Residuals – ARIMA Model Developed for the Time Series of 1ms Intervals. 107

5.6 Time Series Plot on the Entire Test Data – ARIMA Model Developed for the Time Series of 1ms Intervals. 108

5.7 Time Series Plots on a Portion of the Test Data – ARIMA Model Developed for the Time Series of 1ms Intervals. 108

5.8 ACF Plot of Residuals – ARIMA Model Developed for the Time Series of 10ms Intervals. 110

5.9 Time Series Plot of the Entire Test Data – ARIMA Model Developed for the Time Series of 10ms Intervals. 111

5.10 Time Series Plots on a Portion of the Test Data – ARIMA Model Developed for the Time Series of 10ms Intervals. 111

5.11 ACF Plot of Residuals – ARIMA Model Developed for the Time Series of 1s Intervals. 112

5.12 Time Series Plot on the Entire Test Data – ARIMA Model Developed for the Time Series of 1s Intervals. 113

5.13 ACF Plot of Residuals – ANN Model Developed for the Time Series of 1ms Intervals. 115

5.14 Time Series Plot of the Entire Test data – ANN Model Developed for the Time Series of 1ms Intervals. 116

5.15 Time Series Plots on a Portion of the Test Data – ANN Model Developed for the Time Series of 1ms Intervals. 116

5.16 ACF Plot of Residuals – ANN Model Developed For the Time Series of 10ms Intervals. 117

5.17 Time Series Plot on the Entire Test Data – ANN Model Developed for the Time Series of 10ms Intervals. 118

5.18 Time Series Plots on a Portion of the Test Data – ANN Model Developed for the Time Series of 10ms Intervals. 118

5.19 ACF Plot of Residuals – ANN Model Developed for the Time Series of 1s Intervals. 120

5.20 Time Series Plot on the Entire Test Data – ARIMA Model Developed for the Time Series of 1s Intervals. 121

List of Tables

1.1 Flow Table Entry Components – Adapted From [3]. 12

1.2 Flow Table Entry Components – OpenFlow 1.5.1. Adapted From [5]. . 15

1.3 Group Table Components – OpenFlow 1.5.1. Adapted From [5]. 15

1.4 Meter Table Components – OpenFlow 1.5.1. Adapted From [5]. 15

5.1 Results of a Portmanteau Test – ARIMA 1ms Time Series. 107

5.2 Forecasts Accuracy Measures – ARIMA Model for 1ms Interval Time Series. 109

5.3 Results of a Portmanteau Test – ARIMA 10ms Time Series. 110

5.4 Forecasts Accuracy Measures – ARIMA Model Developed for the Time Series of 10ms Intervals. 112

5.5 Results of a Portmanteau Test – ARIMA 1s Time Series. 113

5.6 Forecasts Accuracy Measures – ARIMA Model Developed for the Time Series of 1s Intervals. 113

5.7 Results of a Portmanteau Test – ANN 1ms Time Series. 114

5.8 ANN Forecasts Accuracy Measures – ANN Model Developed for the Time Series of 1ms Intervals. 115

5.9 ANN Forecasts Accuracy Measures – ANN Model Developed for the Time Series of 10ms Intervals. 117

5.10 ANN Forecasts Accuracy Measures – ANN Model Developed for the Time Series of 10ms Intervals. 119

5.11 ANN Forecasts Accuracy Measures – ANN Model Developed for the Time Series of 1s Intervals. 119

5.12 ANN Forecasts Accuracy Measures – ANN Model Developed for the Time Series of 1s Intervals. 120

5.13 Comparisons of Accuracy the ANN and ARIMA Forecasts. 123

ACRONYMS

3GPP	-	Third Generation Partnership Project
A-CPI	-	Application-Controller Plane Interface
AGS	-	Aggregation Switches
API	-	Application Programming Interface
ARIMA	-	Autoregressive Integrated Moving Average Models
ANN	-	Artificial Neural Network
AWS	-	Amazon Web Services
CPI	-	Controller-Plane Interface
CS	-	Core Switches
D-CPI	-	Data-Controller Plane Interface
IMS	-	IP Multimedia Subsystem
IP	-	Internet Protocol
LAN	-	Local Area Network
LXC	-	Linux Containers
MAC	-	Media Access Control
MTC	-	Machine Type Communications
MGE	-	Maximum Goodness-Of-Fit
MLE	-	Maximum Likelihood Estimation
MME	-	Moment Matching Estimation
NBI	-	Northbound Interfaces
NBP	-	National Broadband Plan
NGN	-	Next Generation Network
OFC	-	OpenFlow-Configuration Protocol
OFS	-	OpenFlow-Switch Protocol
ONF	-	Open Network Foundation
OVS	-	Open Virtual Switch
OS	-	Operating System
QoS	-	Quality of Service
SBI	-	Southbound Interfaces
SDN	-	Software-Defined Network
SLA	-	Service Level Agreement
SSH	-	Secure Shell
TCP	-	Transmission Control Protocol
TOR	-	Top of Rack
VLAN	-	Virtual Local Area Network
VM	-	Virtual Machine
VMM	-	Virtual Machine Monitor
WAN	-	Wide Area Network

SYMBOLS

$P(A)$	-	Probability of event A
$f(x)$	-	Probability density function (PDF): $P(a \leq x \leq b) = \int_a^b f(x)dx$
$F(x)$	-	Cumulative distribution function (CDF): $F(x) = P(X \leq x)$
μ	-	mean
$E(X)$	-	Expected value of random variable X
$var(X)$	-	Variance of random variable X
$\bar{x}$	-	Sample mean, defined as $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$
σ_x	-	Standard deviation of random variable X
$\hat{\sigma}$	-	Estimate standard deviation
$\sum$	-	Summation
$\chi^2(x)$	-	Chi-square distribution
df	-	Degrees of freedom
$Y(t)$	-	A time series signal
y_t	-	Values assumed by the time series variable at time t
y'_t	-	A first order differenced time series: $y'_t = y_t - y_{t-1}$
$\sigma(x)$	-	A sigmoid activation function: $\sigma(x) = \frac{1}{1+e^{-x}}$
$S(X)$	-	Skewness of a probability distribution of random varible X.
$K(X)$	-	Kurtosis of a probability distribution of a random variable X.

1 INTRODUCTION

Computer networks consist of various networking devices which are enabled to communicate with each other through physical communication channels (optic fiber cables, copper cables, wireless/free-space etc.) for the exchange of end-user (people, machines or devices) information [10][11]. Due to extensive usage of these communication systems, the past decade has seen diverse and unprecedented changes in Telecommunication (Telecom) and Information Technology (IT). These transformations are observable in the applicable *Telecom infrastructure*, *services* and in the *platforms* through which communication services are consumed. At the core of these advances is the Internet and the protocols which operate on top of it to allow for information to travel around the world.

The research work presented in this report focuses on Software-Defined Networking (SDN). SDN is defined as the physical separation of the control and data planes in networking devices, where a logically centralised controller plane exclusively controls the forwarding behaviour of several data plane devices [1][2]. The communications between the SDN controllers and the data plane devices underpin the feasibility of the physical decoupling of the control plane and data plane of networking devices. In networking devices which tightly couple these planes, the communications path between the control and the data planes is a proprietary "play ground" where only equipment manufacturers and vendors play. The opening of these devices through open interfaces has seen SDN becoming a hot topic in the networking research community and as the next "big thing" in the networking industry.

On the other hand, SDN-based networks face a concern that the SDN controllers are difficult and complex to scale as the size of the network or amount of carried traffic grows. Solutions to design scalable SDN networks do exist in the SDN body of knowledge, but, these solutions do not make use of the statistical properties of the traffic observed by SDN controllers as data packets flow in the data path of the network.

Hence, this research investigates the applicability of the statistical information which can be extracted from the controller plane workload variable in designs of scalable and traffic-aware SDN controllers.

Before delving more into SDN and the research question, two topics are discussed: *Transformation of the Telecoms Network* – in Section 1.1 and *Problems Facing Traditional Internet Protocol (IP) Networks* –in Section 1.2. The purpose of these discussions is to outline the networking challenges which necessitated the concepts of SDN. In Section 1.3, SDN is formally defined and its building bocks are discussed. A brief review of the communications between SDN controllers and data plane devices through the use of the OpenFlow protocol is given in Section 1.4. The problem statement and the research question are given in Section 1.5. The contribution made by this research work into the SDN research body of knowledge is given in Section 1.6. In Section 1.7 a guide to the remainder of the document is presented.

1.1 Transformation of the Telecoms Network

Due to the widespread adoption of packet-switched networks and the proliferation of a myriad of IP-based communication services and devices, a number of IT technologies have been introduced to the Telecom network infrastructure. These include cloud computing; the virtualization technology; IP telephony and the convergence of mobile and fixed line networks through a common IP-based core network (or any packet-switched core network). The convergence of networks through a common IP core also illustrates the merging of IT and existing Telecom networks. An example of an IP-based core network is the IP Multimedia Subsystem (IMS) [12]. Consequently, the resulting network is termed the Next Generation Network (NGN) [13], because it supports a multitude of access (or last mile) network technologies and thereby enabling consumers to access a wide range of communication services.

The communication services whose emergence can be directly attributed to the Internet and other packet switched networks include social networks (e.g. Google+, Twitter, LinkedIn etc.); cloud services (e.g. Dropbox, Amazon Web Services etc.); Internet of Things (IoT) (e.g. smart homes, smart cities etc.); the Quadruple play (voice, data, video and mobility) and others. The Quadruple play was initially called a triple play

(voice, data and video). The "quad" came after the addition of mobile broadband into the pack, which introduced the element of mobility [14]. In part, it is due to these technologies and services that today's society is called a digital one.

Accompanying the emergence of today's digital communities and businesses which rely on IP networks are advances in digital electronics and mobile networks. Electronics has increased access to communication services through the development of electronic devices such as smart phones, laptops, tablet PCs and desktop PCs with improved computing capabilities. The smart phone is seen to be in the forefront of end-user devices which improve access to communication services. This is because the smart phones (and tablet PCs) enable services which were initially consumed through desktop PCs to be accessible on the mobile devices. The smart phone's successes are supported by robust mobile networks including 3G compliant networks (WCDMA, HSPA), 3.9G (LTE, WiMAX) and 4G (LTE-A).

The extensive use of these IT and Communication Technologies (ICT) and services has resulted in the coining of a new term or age in data communications, the "Big Data" age. This is an age where the rate at which data is produced and the volumes of the produced data sets are high to a point that traditional computing and storage facilities are unable to process, analyse store it successfully [15] [16]. Academia and industry have expressed differing views about the definition of Big Data, because of the newness of the concept. A consensus is achieved through a descriptive definition commonly known as the "5Vs" of Big Data [15].

The 5Vs describe the five prominent attributes of the constituents of Big Data [15]: *Volume* – the rate at which data is produced; *Velocity* – the rate at which data is produced and sent around the network; *Variety* – different types of data sets; *Veracity* – data's integrity and *Value* – importance or usefulness of the data. Sources of Big Data are a combination of the aforementioned communication services: social networks, IoT, cloud computing; server virtualization tasks such as Virtual Machine (VM) migration and others. Evidently, Big Data increases the need for the transformation of the networks which carries these services and data sources.

1.2 Problems Facing Traditional IP Networks

Network data volume forecasts indicate that Big Data is expected to get "bigger" with respect to the volume attribute of the 5Vs. It is anticipated that global network traffic volumes will reach 20 Exabytes (10^{18}) by the year 2020 [15]. For this reason, the networking community is compelled to find solutions which will enable networks to cope with the expected demand of these services and devices (i.e. the technologies presented in the preceding section). The most noticeable challenges encountered in current networking are discussed in this section.

1.2.1 Complexity in Network Configuration and Management

Numerous networking devices and software technologies make up traditional communication networks. Current IP networks consist of routers, switches, bridges, gateways, network interface cards, middleboxes etc. Digital packets are handled by all these devices as they travel around the world. The control logic and protocols which facilitate the flow of the packets in the network are distributed in these devices [1]. Furthermore, these devices are manufactured by different vendors around the world and the protocols are defined in isolation by different standardization bodies [1] [17]. Therefore, the network operator is required to configure each device separately using manufacturer-specific commands and tools [1] [17]. Consequently, the network operator faces a complex and lengthy configuration process. In addition, the diversity of the devices in terms of manufacturer, function and protocol makes it difficult to manage the network devices as a single entity.

Network policy definition and deployment becomes difficult when faced with such a plethora of devices. Network operators end up having different management platforms where teams are allocated to manage different products, devices and the services provided through those devices. Hence, network configuration and management is seen as a constant challenge by network operators: on the technical side, it is seen as a complex and lengthy task and on the financial side of the business, it is seen as one of the main causes of inflated network device prices and costly employees. This therefore increases both Operational Expenditure (OPEX) and Capital Expenditure (CAPEX).

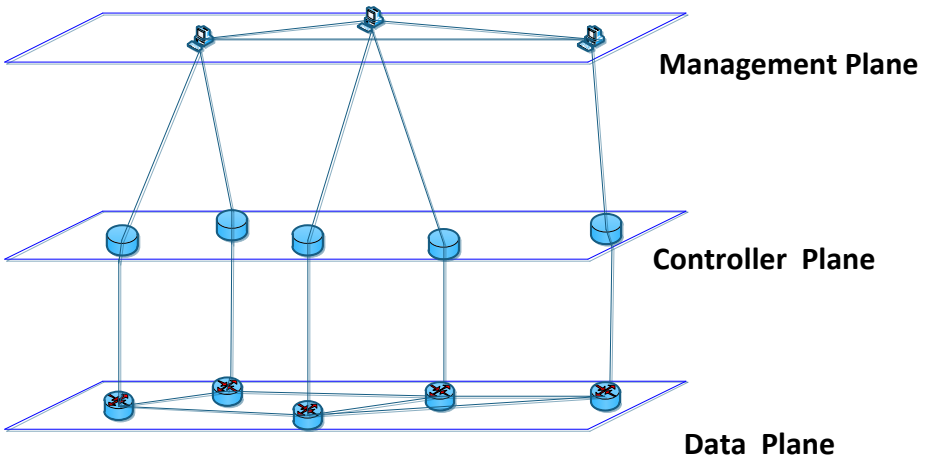


Figure 1.1: An Illustration of a Network Devices's Functional Planes – Adapted From [1].

1.2.2 Vertical Integration in Networking Devices

Networking devices operate with three functional planes: the management plane, the control plane and the data plane [1]. In traditional IP networking devices such as routers, the control plane and the data plane are embedded in the same device. This is called vertical integration in the networking device [1] [15]. Figure 1.1 illustrates a layered view of a networking device's functional planes.

The control plane hosts the software and protocols which are used for controlling the routing of network traffic in the data plane. This is done through populating the forwarding tables of the data plane elements with traffic routing instructions [1].

The management plane manages the control functionality of the data plane devices remotely. It defines the network policy for traffic forwarding, apply it to the controller plane and then monitor the control functionality to ensure that the correct policy is followed [1].

The data Plane is also called the data forwarding plane because it is composed of the devices which are responsible for forwarding data in accordance with the defined network policy.

In traditional IP networking where the network employs vertically integrated devices, the resulting network is decentralized [1]. This is because the control and management

of the network is distributed over a number of devices which are located at various locations of the network. Highly decentralized and distributed systems have been used effectively in the past. For example, these systems are the back-bone of the Internet, Ethernet and other packet-based networks which are known to have driven the digitalization of today's society thus far [1]. In the early days of IP networking, network resilience and performance were major design goals, these were effectively achieved through the decentralized structure of IP networks [1].

On the other hand, due to the vertical integration; isolated definition of protocols; the use of vendor-specific commands in network configuration and management platforms among others, current networks are complex, costly to operate and difficult to improve and manage [1] [17] [15]. It is therefore essential to point out that network engineering and innovation for the improvement of existing networking services and introduction of new ones has been brought to a halt due to these challenges in traditional IP networking [1] [17]. In light of the need to confront these challenges, SDN brings a paradigm shift in networking: First to the design of the devices which make up the network, and second, to the design of the network itself. SDN is not the ultimate solution to all these problems, but it provides a platform through which these problems can be confronted.

1.3 The SDN Promise

Software-Defined Networking (SDN) is defined as the physical separation of the network devices' control plane and the data plane [1] [17]. SDN is set apart from past networking approaches by two defining attributes: First, the physical decoupling of the control plane from the data forwarding plane in networking devices such as routers and switches. Second, centralization of the control software which enables a single software-defined entity to exercise the network's control functionality over a number of data plane elements [18].

The control software and the control devices are not necessarily physically centralized [18]. In most SDN-based network deployments, these devices are logically or physically distributed. However, the software running in these devices is the same and keeps the same state of the network resources under its controls. It is through this attribute of the SDN architecture that a logically centralized control platform is achievable. This

architecture results in the data plane elements such as switches and routers becoming "passive" or "simple" forwarding devices which do not have intelligence as in previous architectures [1]. An SDN controller which represents the control plane (also called the controller plane) of a networking device has complete control over the data plane elements (e.g. routers, switches etc.) and the networking resources they (data plane elements) expose. Network resources exposed by data plane elements include forwarding/routing tables and the state of the devices (e.g. the queue lengths in switch buffers) [2] [18]. Through these resources, SDN controllers are enabled to manage and control the behaviour of data plane elements.

1.3.1 SDN Architecture and Principle

The idea behind SDN is to improve data communication networks by providing open interfaces, which are also called Application Programming Interfaces (APIs). The objective of these interfaces is to make it feasible to develop software that can perform at least the following functions in a network [2]:

- Control the functionality provided by individual networking devices (i.e. routers, switches, middleboxes etc.) as well as the connectivity provided by a group (or groups) of these network resources.
- Control the flow of user traffic through the network resources at various granularities, that is, per packet or groups of packets selected by a specific criterion (e.g. a flow).
- Allow for inspection and modification of traffic carried in the network in real time, i.e. while the network is running user applications.

The open APIs provided by the SDN architecture are explained with reference to Figure 1.2, which shows a simplified view of the SDN architecture [17] [2].

Figure 1.2 shows two controller plane interfaces: *Data and Controller Plane Interface (D-CPI)* and *Application and Controller Plane Interface (A-CPI)*. Across the D-CPI, the controller plane controls and monitors data plane resources. The A-CPI allows SDN applications running on the application plane to receive services made available

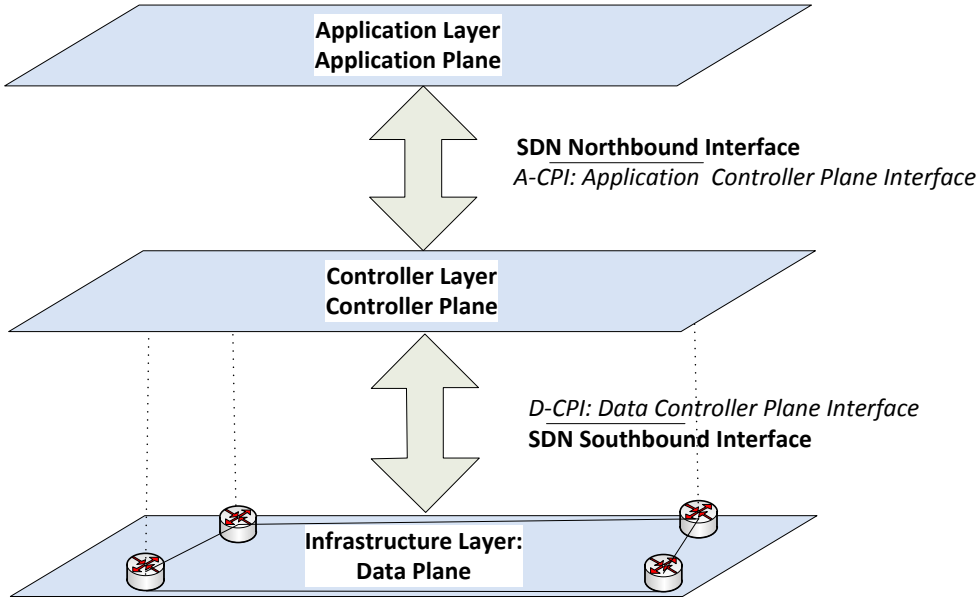


Figure 1.2: The Three Layered SDN Architecture – Adapted From [2].

by SDN controllers [2]. The applications rely on resources exposed by the controller plane to perform network functions. Network policy deployment and management tasks are handled by the controller plane through these interfaces.

The A-CPI and D-CPI interfaces are widely known as *northbound* interfaces (**NBIs**) and *southbound* interfaces (**SBIs**) respectively. The core concept of SDN, which is the decoupling of the controller plane from the data plane is managed through the SBI. The widely used protocol in the SBI is OpenFlow [17]. In the literature, it is almost impossible to read through an SDN discussion without coming across reference to the OpenFlow protocol. The term SDN is known to have been coined from work around OpenFlow at Stanford University [1]. Therefore, OpenFlow is seen as a major enabler of the SDN concept.

SDN advocates for open networking interfaces (in the NBIs and SBIs) through which networks can be programmable. In efforts to develop common standards for the SDN architecture and promote its adoption, a number of industry led organizations were established, the early ones are Open Networking Foundation (ONF) [19] and the Open Daylight Project [20] [18]. The three layered SDN Architecture shown in Figure 1.2 is standardized by the ONF. This architecture is selected to be used in this research because the ONF plays a leading role in the development of OpenFlow-based SDN

SBIs. OpenFlow is also preferred for the study presented in this report, because it is as old as the first SDN controller and for this reason it is considered the most mature SDN SBI protocol.

The SDN architecture defines an *Information model* which is used to facilitate the manipulation of network resources through the open interfaces [2]. An information model is defined as [2]:

"*A set of entities, together with their attributes and operations that can be performed on the entities, an information model instance is available at an interface*"

The information model instance is synonymous to an object in an object-oriented programming language. The APIs which make it possible to program the different network resources are derived from the applicable model instances. For example, in a situation where the SDN controller needs to modify a forwarding table of a network element, the D-CPI will present the resources (i.e. the forwarding table) of the network element as an information model instance to the controller. The communication protocols which are used in the NBIs and SBIs follows the concept of an information model instance.

1.3.2 Benefits of SDN

Through the SDN architecture and model, the following benefits of SDN can be pointed out [17] [1].

1. **Logically Centralized Intelligence** comes as a direct consequence of the SDN core concepts: the physical decoupling of the control plane and the data plane; and the consolidation of the control software applications. The logically centralized controllers make it feasible to apply control and management tasks to the network based on a global view of the network resources. This is impossible in traditional IP networks, because current network devices are not aware of the state of the entire network. Therefore, it can be undoubtedly stated that optimal traffic routing is easier in SDN networks which are equipped with a network-wide knowledge of the state of network devices compared to traditional network in which connected devices are only aware of the state of devices directly connected to them.

2. **Programmability** is a highly sought attribute in computer networks. The open interfaces which the SDN architecture prescribes are the key enablers of network device programmability. For example, OpenFlow enables both equipment manufactures and network operators to develop software which can be used to control the network resources exposed by the data plane. This allows for easy development of new network applications and encourages innovation that is not limited to vendor specific protocols and proprietary software tools. Moreover, programming through open interfaces makes it easy for different vendors to create network devices which are compatible with each other, thereby providing a solution to the long standing interoperability problem among network devices from different manufactures.

3. **Abstraction** is a widely used concept in computer science and software engineering. In SDN, abstraction means the presentation and dedication of network resources and their state to network applications and other devices [2]. The exposure of the network resources through abstraction (or virtualization depending on the criteria of abstraction) provides an environment where network applications can be developed independent of the underlying network equipment or hardware. As a result, the applications developed and the control software can be moved from one networking equipment to another with ease. Therefore, abstraction also enables software or application portability.

4. **Flow-based Forwarding** is where the routing of traffic in the data plane devices is done on a flow basis. The packets of a flow receive similar network policies in the data plane. This capability allows for granular traffic forwarding where different policies can be specified at session, user, device and application level [17].

SDN, therefore, provides a platform through which networking engineers in academia and industry can begin to confront the challenges which have faced computer networking for years. Some of these challenges have been outlined in the preceding sections.

1.4 State of the Art: Southbound Interface in SDN

The SDN components which encompasses the southbound interface on both sides are the controller plane and the data plane. Shown in Figure 1.3 is the SDN architecture without the application plane. This is done to zoom into the building blocks of the controller and data plane: the switches representing the data plane elements, the southbound interface (control channel) and the controller plane.

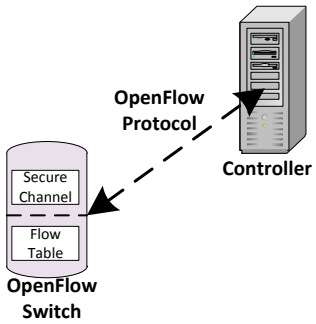


Figure 1.3: OpenFlow 1.1.0 Switch Components – Adapted From [3].

1.4.1 Data plane

The data plane comprises the networking equipment (e.g. routers and switches) in a similar way to the setup in traditional IP networks. In contrast to data forwarding devices found in traditional networks, the devices making up the SDN's infrastructure layer are simple devices which do not have intelligence [1]. The intelligence is moved to the controller plane. The central attribute of the data plane element is the capability of traffic forwarding. Therefore, a data plane element can be a software or hardware entity [1]. It is this attribute of the data plane resources which makes it possible for a device in the data plane to act as a router, a switch or a firewall. This is feasible because the SDN application through the controller plane controls the routing instruction contained in the forwarding tables of the data plane elements. Furthermore, these resources are designed to be compatible with the protocols of the SBI, e.g. OpenFlow compliant switches.

1.4.2 Southbound Interface Protocol: OpenFlow

OpenFlow is the most dominant southbound interface protocol in SDN networks [1]. The first OpenFlow standard and switch specification guideline was released for use in commercial switches in December 2009 (OpenFlow 1.0.0) [3]. The OpenFlow networking architecture comprises three components: OpenFlow compliant switches, a secured controller-to-switches channel and an SDN controller that is also compatible with the OpenFlow protocol. The switches represent the data plane; the secure channel represent a southbound interface path and the controller represent the SDN control plane which is decoupled from the data plane. An illustration of the OpenFlow architecture is shown in Figure 1.3 [3]. This Figure presents OpenFlow 1.0.0, where the OpenFlow switches consist of two components: The first component is a flow table which is used for packet lookup, classification into flows and forwarding in accordance to installed policy [3]. The second component is an OpenFlow Secure channel to communicate to a remote SDN controller. In this specification, the support for multiple controllers was not defined [3].

Table 1.1: Flow Table Entry Components – Adapted From [3].

Header Fields	**Counters**	**Action**

A flow table hosts flow table entries or flow rules. These are packet forwarding instructions used to control the flow of traffic in the OpenFlow-based SDN network. A flow table entry contains three fields: *Header Fields*, *Counters* and *Actions*. These fields are depicted in Table 1.1.

An incoming packet's header is parsed and compared against the *header fields* of the flow table's flow entries to determine an applicable flow table entry. Once a match is found, the corresponding *counters* are updated and the *actions* applicable to the matched flow entry are applied to the packet. The packet processing mechanism followed in OpenFlow 1.0.0 is shown in Figure 1.4.

The OpenFlow protocol has improved tremendously in the last five years (December 2009 - December 2015), with 13 switch specification updates published. The latest specification as of this writing is OpenFlow 1.5.1. which was published in April 2015

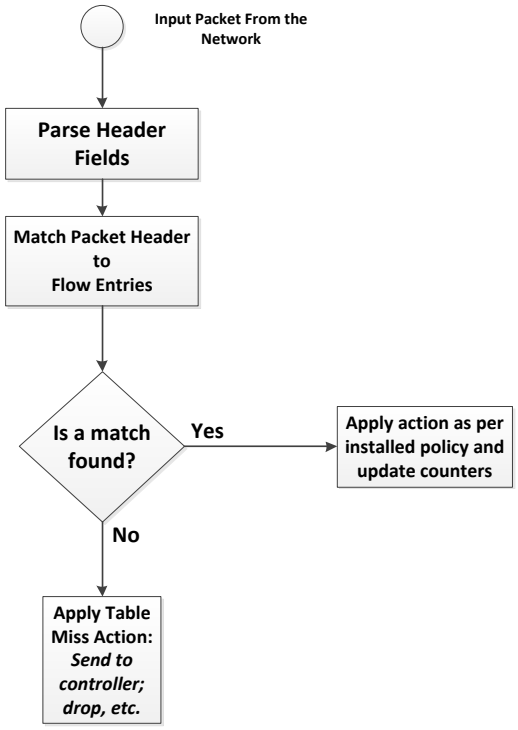


Figure 1.4: OpenFlow Basic Packet Matching Process – Adapted From [4].

[21]. However, the high level principle of the protocol (Figure 1.4) has not significantly changed. Packet lookups are still performed to determine an incoming packet's flow, packets are still forwarded according to rules contained in the flow tables entries and communications between the switches and the controllers still take place to facilitate the actions taken on incoming packets. For this reason, the ONF refers to a switch which implements OpenFlow 1.0.0 as a basic OpenFlow compliant switch. This means that in its basic form, an OpenFlow switch will have at least one *flow table*, one *controller-to-switch channel* and one *controller*.

In efforts to bring to light the current capability of the protocol and a clear route to the research question, the features of the protocol which are of interest in this work are briefly discussed. These are packet processing mechanism – *Pipeline processing*; support for multiple SDN controllers – *The Secure OpenFlow Channel*; and forwarding capabilities obtained through the use of *Group Tables* and *Meter Tables*. Figure 1.5 shows the OpenFlow switch components as postulated in OpenFlow 1.5.1 [5].

Pipeline Processing: The use of multiple flow tables for packet lookups and traffic

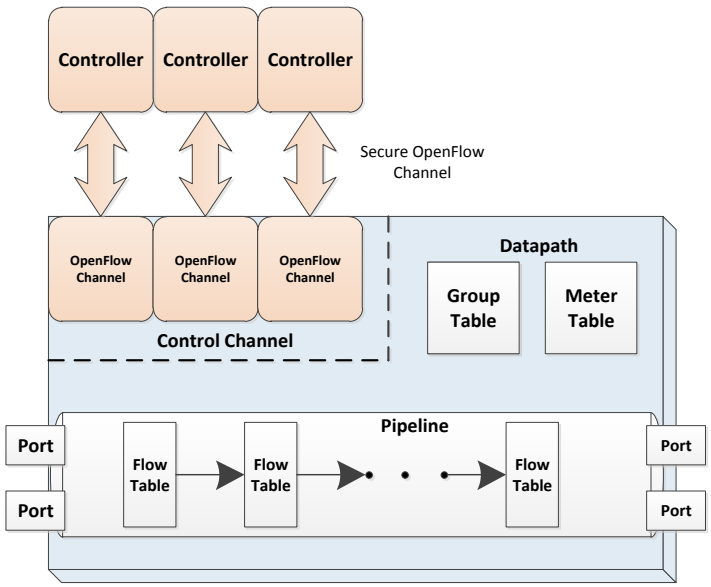


Figure 1.5: OpenFlow 1.5.1 Switch Components – Adapted From [5].

forwarding was first introduced in OpenFlow 1.1.0, which was released in February 2011 [4] [5]. The flow tables still contain flow table entries in the same manner as those in the basic OpenFlow protocol (OpenFlow 1.0.0). In the basic OpenFlow switch, packets are processed through the use of a single flow table whereas in the improved OpenFlow (OpenFlow 1.1 and above) protocol more than one flow table may be used to process packets. The pipeline processing defines how an incoming packet is processed by multiple tables. The process is as follows [5]: When a packet is received from the network, it is matched against the header fields of an ingress flow table, if a matching flow table entry is found, then an instruction contained in the flow table entry is executed. This instruction determines the next activity which will be performed on the packet.

The outcome of the instruction may direct the packet to another flow table which will once again repeat the same process or send the packet direct to an egress port. This process entails that instructions to be performed on the packet can be accumulated over the entire pipeline, where a metadata field is used to send instruction between flow tables. The pipeline process enables OpenFlow to support complex packet forwarding tasks and significantly improves the basic OpenFlow switch [4]. Table 1.2 [5] shows the fields of a flow table entry which suports the pipeline processing. The obvious difference between the flow table entries of a switch with one flow table and the one

with multiple flow tables is the *Instruction* field which replaces the *Action* field in the former.

Table 1.2: Flow Table Entry Components – OpenFlow 1.5.1. Adapted From [5].

Match Fields	**Priority**	**Counters**	**Instructions**	**Timeouts**	**Cookie**	**Flags**

Group Table: The group table was also introduced in OpenFlow 1.1.0, it enables the manipulation of groups of flows [4]. Flow tables concentrate on individual flows and group tables on sets of flows, which consequently allows for traffic to be controlled at variable granularities (number of packet affected by an instruction). The components of a group table are: *Group Identifier*; *Group Type*; *Counters*; and *Action Buckets*, these are shown in Table 1.3 [5]. The Group Identifier is used to identify the group; the group type defines the behaviour of the flows of the group and the counters and the Action Buckets have the same functions as the counter and action of a flow table entry (explained earlier - Table 1.1) respectively.

Table 1.3: Group Table Components – OpenFlow 1.5.1. Adapted From [5].

Group Identifier	**Group Type**	**Counters**	**Action buckets**

Meter Table: The meter table is another improvement to the OpenFlow traffic forwarding capability, it was introduced in OpenFlow 1.3 [4]. It allows the protocol to monitor and control Quality of Service (QoS) tasks such as rate-limiting to a flow or a group of flows [5]. The meter tables are attached to flow entries in order to monitor and modify the bandwidths of the applicable flows accordingly. Main components of the meter table are shown in Table 1.4 [5].

Table 1.4: Meter Table Components – OpenFlow 1.5.1. Adapted From [5].

Meter Identifier	**Meter Bands**	**Counters**

1.4.3 Secure OpenFlow Channel

The secure channel between the controller plane and the data plane in an OpenFlow based SDN is the only interface through which the SDN controller plane (also called the network's operating system (NOS) [1]) configures, controls and manages the data plane elements [5]. The channel supports the use of both single and multiple controllers. The OpenFlow protocol defines three groups of messages or signals with which the switches and controllers communicate. The three messages are: *Controller-to-Switch*; *asynchronous* and *symmetric* messages [3] [5].

1. **Controller-to-Switch:** These messages are initiated by the controller plane and their purpose is to control and monitor the switches [5]. These messages have sub-types, which include: `Features`, used for switch identifications in terms of network identity and capabilities; `Modify-State`, used to manage the state of the switches.

2. **Asynchronous:** These are messages sent by the data plane elements to the controller without the controller's solicitation [5]. These messages are used to alert the controller of the data plane element's state such as packets arrival. These messages give over a packet's control to the controller plane device. The main asynchronous messages include `packet in`, which the data plane uses to inform the controller of events such as a table-miss; `flow-removed`, which is used to tell the controller about expired flows. The message of utmost importance in this research is the that of `packet in`, because it represents the demand of the data plane on the controller's resources.

3. **Symmetric:** These are messages which may be initiated by either the controller plane or the data plane without solicitation from either side [3]. Symmetric messages are used to keep the communications channel live. Subtype messages include `Hello` messages which are exchanged between switches at the start of a connection; `Echo` messages, which are used for verification of the liveness of the communications channel.

1.5 The Research Question

One of the pillars upon which the SDN concept rests is *flow-based* traffic forwarding. This traffic forwarding technique opens a platform for easier traffic control, where policies can be applied to groups of packets. In SDN-based networks, a flow can be broadly defined as a sequence of packets between a source and a destination [1]. The OpenFlow specification uses a flow table entry such that the values of its header or match fields define a flow, i.e. packets with similar header fields belong to the same flow. Two methods or design choices can be followed for handling the installation of flow table entries in OpenFlow switches. These are classified as *reactive-forwarding* or *proactive-forwarding* [4].

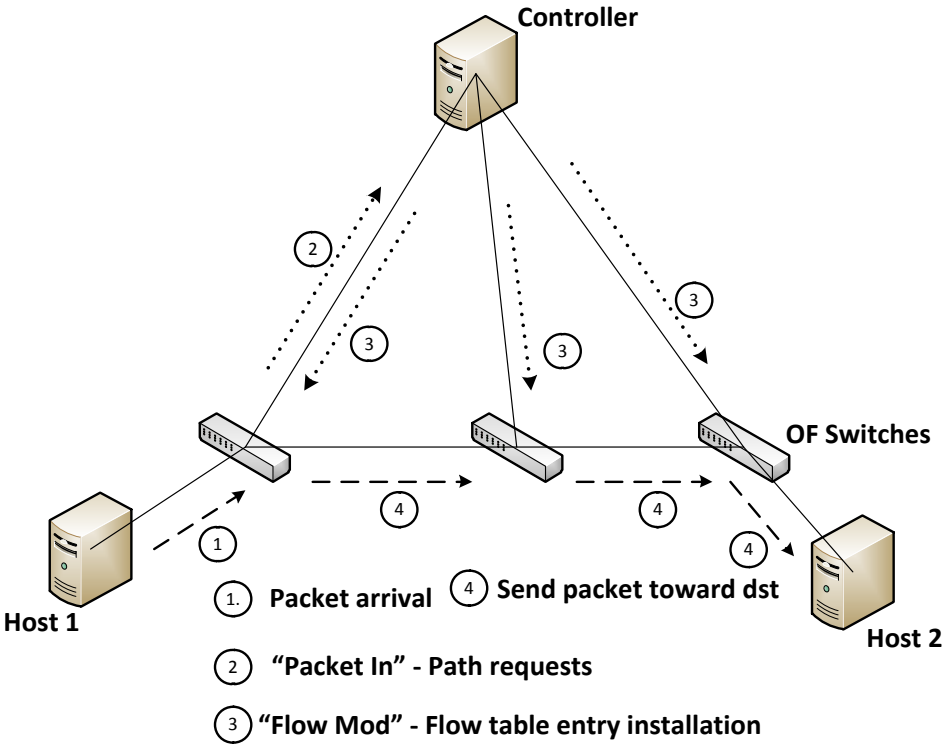


Figure 1.6: Reactive Forwarding in OpenFlow Switches – Adapted From [4].

Proactive-Forwading: In this method, the flow table entries or rules for forwarding packets are defined in advance and installed at the switches. This method assumes that all packets to be received by the SDN switches are known in advance, or, not that the packets are known, but the rules to handle all incoming packets are known. In efforts to reduce the packet lookup time in the switch, the flow table entries are generally stored in the switch's ternary content addressable memory (TCAM). The TCAM is computationally expensive and power intensive, and for this reason, it is

kept reasonably small. The limited TCAM presents a major drawback in the use of proactive forwarding, because as the network traffic grows, more granular flow matches may be required, which in turn may demand a large number of flow table entries [4]. The pipeline of flow tables introduced in OpenFlow 1.1.0 is an attempt to increase the number flow tables in a switch. However, the switch's storage and computing resources are finite and sometimes limited (i.e. the TCAM is kept small), therefore other methods for managing flow table entries or flow initiations are still sought.

Reactive-forwarding: This method defines the *table-miss*[1] flow table entry such that a packet without a match is sent to the controller, which then decides on the action to be performed on the packet. One of the actions may be to add a flow table entry to the flow tables of the switch which received the unmatched packet and to the switches in the path that the packet needs to traverse to get to its destination. The reactive forwarding method is shown in Figure 1.6 [4]. This method of packet handling in SDN switches takes place even in proactive forwarding designs, these are designs where the "table-miss" flow entry is defined such that it sends unmatched packets to the controller for further processing. Consequently, it is generally accepted that the sending of unmatched packets to the controller is the way OpenFlow based SDN functions.

The Big Data era of today and the accompanying changes in network traffic patterns entails that Wide Area Networks (WANs) and Data Centers (DCs) will continue to experience increasing traffic volumes. The wide-range of traffic types (voice, video, data and mobility) require different QoS settings, and therefore, demand more fine grained matches on the flow table entries. Figure 1.7 shows the queues a packet may have to join before getting a service in an OpenFlow switch: Firstly, the packet is enqueued in a switch while awaiting the result of a flow table entry lookup. Secondly, if the outcome is a table-miss flow entry, then the same packet may be sent to the controller where it will join another queue. In other words, the controller plane becomes a bottle-neck in an event of a large number of new flows or table-miss events. This is generally known as the controller plane scalability challenge or congestion of the controller plane.

Differing solutions have been proposed in both the controller plane and the data plane

[1]The purpose of a flow-miss table entry is to handle unmatched packets. It features in all flow tables by default.

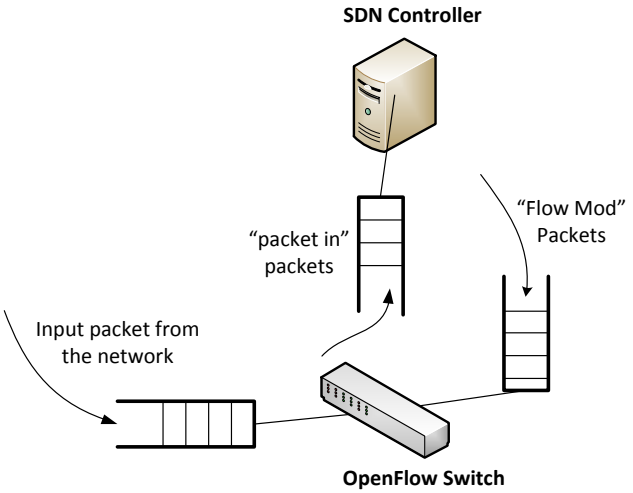


Figure 1.7: OpenFlow Basic Packet Matching Process.

for averting a congestion situation in the controller plane. About four solution attempts can be linked to the data plane: Firstly, an increased number of flow tables (i.e. the flow table pipeline) in a switch. Secondly, installing flow table entries in the switch's TCAM. Thirdly, the use of wildcards in the flow table entries header fields (in some instances compressed wildcards). Lastly, moving some controller plane functions to the switches.

The solutions which concentrate on the use of multiple flow tables or compressed flow table entries turn to be inadequate, because the TCAM and the other switch's resources are finite and limited. The solutions which move functions of the controller plane to the data plane is also questionable, because the extent of network intelligence or control functionality that can be moved to the data plane is not known. The SDN architecture [2], admits that some decision making or network intelligence should be a responsibility of the data plane elements, but it does not clearly define the functions of the control plane which can be given to the data plane. In other words, this is left as an open ended discussion. On the other hand, the controller plane is given the right to exercise exclusive control over the set of data plane resources it controls. For this reason, the research path presented in this document concentrates on controller plane solutions.

The solutions which are based on the controller plane can be classified into three distinguishable classes: distributed controller plane instances; hierarchical controller planes and multi-threaded controller software [22]. In the distributed controller plane class,

the controllers maintain the same network-wide view through the use of a controller-to-controller communication system. The consistent network knowledge which is shared by the physically distributed controllers results in the *logically-centralized* controller plane. The hierarchical controller planes use different levels of the hierarchy for different control functions. For an example, a hierarchy with two levels of controllers may use the root level controllers to maintain a global view of the network and use the other level of controllers for performing localized control task to a subset of data plane elements [22]. The multi-threaded controllers exploit the parallel processing capability of computer facilities with multiple processing cores. In this setup, threads can be scheduled to run concurrently and thereby improving the packet processing power of the controller plane.

The communications channel between the control plane and the data plane in networking devices has been a proprietary "play ground" since the inception of computer networking. The advent of SDN as a networking architecture that separates the two planes has opened access to this play ground. Studies or solutions which attempts to avert a situation wherein the controller plane becomes a bottle-neck have been developed in isolation with those that monitor network traffic and the traffic's demands on the network. For example, the distributed controller plane solutions do not rely on the data plane to controller plane network traffic statistics to determine an optimal number of distributed controller instances.

The *controller-plane* oriented solutions (distributed controllers, hierarchical controller etc.) which attempt to prevent the possible controller plane congestion in SDN-based networks lack the use to data plane to controller plane traffic statistics. More importantly, the traffic between the controller plane and the data plane has not been extensively studied and applied in the design of SDN controllers (software or hardware) and controller plane networks (distributed or hierarchical designs). As a result, the objective of the research work presented by this report is to bring the traffic statistic between the control plane and the data plane into use in designing of controller planes. Therefore the research question is formulated as follows:

The current controller plane workload balancing schemes which are used to avert a situation where the controller plane suffers from congestion lack the use of network traffic statistics. In order to improve these solutions, this research investigates the use

of time series statistical models to forecast the controller plane workload. It is believed that the forecasts can be used in the designs of controller planes or in improving existing controller plane load balancing applications. The following two questions are explored:

1. *What are the statistical distributions of the traffic between the controller plane and the data plane? The data is fitted to standard distributions such as log-normal, Poisson etc. to see if currently followed models are applicable in this setup.*

2. *Which Statistical models forecast the network traffic reasonably well to perform load balancing tasks such as resource allocation at the controller plane? Explored Statistical forecasting methods are Auto Regressive Integrated Moving Average Models (ARIMA) and Feed Forward Artificial Neural Networks (ANN).*

The SDN literature observed does not provide evidence of the use of traffic statistics (especially data-plane to controller-plane) in designs of controller plane platforms or in aiding their performance. The lack of data plane to controller plane traffic awareness in the controller plane designs means that it is not known if a given controller will be able carry out all the computations the data plane demands on it without going into a state of congestion or into a state that could degrade the QoS of user applications. Thus, this research aims at exploring statistical characteristics of this traffic. It is believed that knowledge of the nature of the traffic seen by SDN controllers can aid the designs of SDN controllers and help in the alleviation of congestion of SDN controllers.

1.6 Contribution of the Research

The above-mentioned research questions are aimed at achieving two major outcomes: Firstly, the characteristics of the arrival process of the traffic seen by SDN controllers are investigated. Secondly, the feasibility of representing the controller workload as forecasts in the network-wide view data structure of SDN controllers is also investigated. In efforts to produce credible results, a realistic network prototype is developed by making use of the Mininet [23] network emulator and ONOS [24], an open source SDN controller. The details of the network prototype and its applicability in this research are given in Chapter 3.

The first outcome of this research is given in Chapter 4. The controller workload signal

is modelled as an arrival process, where the inter-arrival times and arrival rates of jobs to be executed by the controller plane are investigated. The results reveal that the inter-arrival times of the controller workload packets do *not* follow the widely used Markov Arrival Process (MAP) such as the $M/M/1$ queueing system. This is because the inter-arrival times were found to follow a beta distribution instead of the negative exponential distribution. A similar conclusion has been made in research conducted by the 3GPP in Machine Type Communications (MTC) [25]. At the time of the writing of this research report, there was no study that was found to have reported that the beta distribution is the most applicable probability distribution to model the inter-arrival times of `packet in` messages in an SDN controller. The statistical properties of the beta distribution which make it applicable in modelling the inter-arrival times of the observed traffic patterns are discussed in Chapter 4. About 1-2% of the flows were found have inter-arrival times less than 10μs and more than 80% were found to have inter-arrival times in the range of 10μs–10ms. This finding is consistent with arrivals of flows recorded in data center networks reported in [26], [27] and [28].

The second outcome of this research is presented in Chapter 5. This outcome aims to establish the feasibility of forecastng the controller workload signal by making use of Autoregressive Integrated Moving Average Models (ARIMA) and Artificial Neural Networks (ANN). These forecasting models were applied to the time series representations of the controller workload which was sampled at three different time intervals: 1ms, 10ms and 1s. It was found that the time series data with 1ms and 10ms intervals was discretely valued and its forecasts can be best obtained by making use of INteger valued AutoRegressive (INAR) models instead of ARIMA or ANN. However, the ANN model was found to be applicable in forecasting of the time series data captured at 10ms. Both ARIMA and ANN proved to be capable of forecasting the time series of 1s time intervals. The forecasting results established that statistical forecasting techniques can be used to forecast the workload seen by SDN controllers. It was also realised that these forecasts can be used to detect the need to increase controller plane instances in a distributed controller platform or the number of worker threads in multi-threaded controllers.

1.7 Structure of the Report

The remainder of this report is as follows: A literature review of controller plane oriented solutions to the SDN scalability problem is presented in Chapter 2. In Chapter 3, the network prototype used in the research is discussed. The investigation into the statistical characteristics of the controller workload variable is given in Chapter 4. Approaches to forecasting the controller workload are given in Chapter 5. A conclusion is given in Chapter 6.

2 LITERATURE SURVEY

A novel networks design approach was introduced by the 4D project [6] in 2005, which was prior to the invention of SDN as paradigm shift in networking. This is one of several attempts by the networking research community to confront the fragility and complexity of traditional IP and Ethernet networks. The 4D project pointed out that the instability and complexity of these networks is due to the tight coupling of the network's decision logic and the distributed protocols in networking devices [6] (i.e.*vertical integration* in networking devices) in the preceding chapter. In light of the then state of networking devices, the 4D project postulated the "4Ds" approach to networking: *Decision*, *Dissemination*, *Discovery* and *Data*. These are governed by three defining principles: *Network-level Objectives*; *Network-Wide View* and *Direct Control*.

Network-level objectives: A network has to satisfy a number of objectives which depend on the network's applications and end-users. Therefore, one network will have different performance objectives compared to another. The *network-level objectives* principle recommends that the configuration and control of the network which enables it to achieve its performance requirements should be expressed separately from the network elements. Furthermore, the configurations need to be done in high-level commands which are generic and not vendor or equipment specific [6]. This principle is achieved through the abstraction feature (see Section 1.3.2) of SDN controller planes.

Network-wide view: It is believed that a robust network requires a network-wide view of the state of the network's resources. The network-wide view means a consistent representation of the networking devices as well as their states, that is, network topology, traffic statistics and crucial network events [6]. This representation of the network should be kept in a centralized system where the decision making plane can access it. This is the notion of a logically-centralised controller plane in SDN-based networks.

Direct-control: This principle talks to the complete removal of traffic forwarding control from data plane elements. The application of this principle suggests that traffic routing and forwarding in data plane forwarding tables must be exclusively performed by control and management planes which are not embedded in the same device as the data plane elements [6]. This is consistent with the physical separation of the data plane and the forwarding plane in networking devices as also recommended by the SDN concept.

These principles of the 4D project show that the SDN concept stems out from the 4D project. The original architecture of the 4D design shown in Figure 2.1 [6] has been modified to include features of SDN. The 4D project also pointed out the challenge of scalability in networks which are centrally controlled. This was in regard to the observation that traditional networks consist of a multitude of distributed networking devices which also have the packet forwarding protocols embedded in them. Hence, the scalability challenge came from the desire to achieve the *Network-wide view* and the *Direct-control* principles in a network with thousands of networking devices.

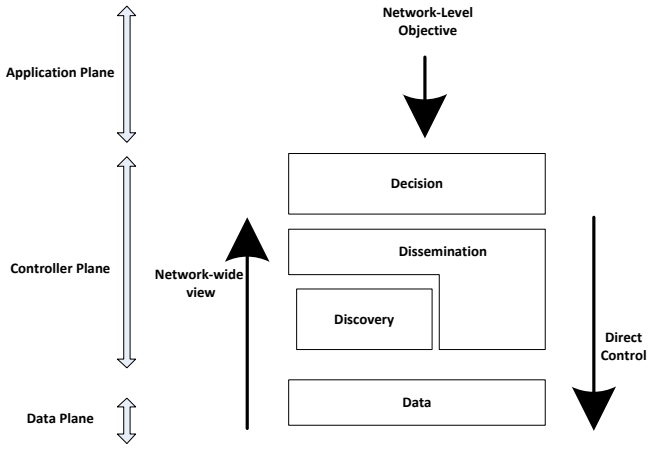


Figure 2.1: The 4D Architecture Divided to Indicate the 3 Layers of the SDN Concept – Adapted From [6].

The challenge of network scalability is associable with exploring the following questions:

- At what network scale (which includes the number of all network resources and users) can a network-wide view be achieved and be usable to make decisions to control an entire network?

- Is it feasible to maintain a *network-wide view* and *direct-control* for networks as large as those seen in current IP-based architectures?

In SDN, the controller plane has the responsibility to keep a coherent network-wide view. Applications running on top of the controller plane use the network-wide view to manage and control the traffic forwarding behaviour of data plane elements. This research focuses on the controller plane designs which are used to address this scalability challenge. In particular, SDN networks which uses the OpenFlow protocol as a southbound interface are considered.

The idea behind the research work presented in this report is to determine the usability of the data-plane to controller-plane traffic statistics in designs of controller plane software. It is believed that the traffic statistics can help to avert a situation where the logically centralized controller platform is not scalable enough to meet today's network needs.

The literature survey of existing SDN controller plane solutions reviewed are: Section 2.1 discusses the designs of existing SDN controllers. Section 2.2 explains the design approaches followed by distributed and hierarchical controllers. Section 2.3 presents software tools which are used in measuring of the performance of SDN controllers. Section 2.4 discusses methods which are employed in the analysis of network traffic in SDN-based networks. Section 2.5 presents traffic engineering solutions adopted in SDN. A summary of this chapter is presented in Section 2.6.

2.1 SDN Controllers

Several SDN controllers have been developed since the inception of the OpenFlow protocol in December 2009. OpenFlow is the most prevalent southbound interface protocol and is supported by a large number (almost all open source SDN controllers) of SDN controllers [1] [29]. For the purpose of this research report, only OpenFlow-compliant SDN controllers are discussed. These include NOX, Maestro, Beacon, Floodlight, and Trema. Other controllers are discussed under distributed and hierarchical controller platforms in Section 2.2. These discussions intend to explore the general approach followed in controller designs with particular emphasis on network scalability and controller plane workload balancing.

2.1.1 An Operating System for Networks (NOX)

In light of the novel networks design approach postulated by the 4D project, the first open source SDN controller called **NOX** [30] was developed. NOX is a C++ based SDN controller that employs a computer operating system approach to the problem of network control and management [30]. The authors of the NOX paper observed that early computers did not have generic programming abstractions for the underlying hardware. For this reason, computer programs were written in machine language [30]. This presented a problem, because computer software applications written in low-level languages are susceptible to bugs and the resulting code is hard to debug [30].

The advent of computer operating systems came with the idea of generic abstractions and interfaces which can be used to access a computer's hardware resources by making use of high level programming languages. The idea behind NOX as an SDN controller lies on the use of a similar approach, that is, an approach of an operating system for networks. The network's operating system is expected to provide open interfaces which can be used to access the network's resources from a centralized point in the network [30]. It is through these interfaces that network applications may be developed to control and manage the network [30].

The main components of the NOX controller platform include:

- A NOX controller - a processing unit hosting the NOX software.
- A database system containing the *network-wide view*.
- A minimum of one server on which the NOX software and the network applications run (i.e. control and management applications).

Several controller processes run on the servers in such a way that they are evenly distributed on available servers [30]. These processes use the same network-wide view to get the state of network resources [30]. The network view includes switch-level topology, user locations, hosts and others. It is important to note that the network view *does not* include the network's current traffic [30]. In this setup, a trade-off is made between the amount of information making up the network view and the scalability of the network [30]. In case the network's traffic was included, the controller would face

the daunting task of keeping a coherent traffic report for all connected devices. This is difficult, because the larger the number of devices connected to the network, the faster the traffic changes, which means more workload for the controller plane to keep a consistent network-wide view.

Three primary computational tasks are processed by the NOX controller [30]: *packet arrivals*; *installation of new flows* and *updating of the network view*. Due to all these processes running on the servers, the NOX paper argues that the processing capability of the controller plane can be increased by adding processing servers. However, there is no technique or approach that NOX employs to determine the optimal number of servers which may be required to avert congestion of the controller. Hence, increasing processing servers is considered inadequate to combat the controller scalability challenge. Furthermore, the NOX paper does not indicate that network traffic statistics may be applied to improve the scaling or processing capability of the NOX-based controller plane. As a result, this leaves a gap for a detailed study of the traffic patterns observed by the controller plane.

2.1.2 Maestro

Maestro is a Java-based multi-threaded SDN controller that was developed at the Rice University in parallel with development of the NOX controller [30] [31]. This controller was designed to address the scalability challenge to which the NOX controller may be susceptible. Maestro employs parallelism in multi-cored computing technologies. This is done through the use of multiple threads in the controller software as opposed to the single threaded software used in NOX [31]. The design objective of the Maestro software in terms of threads management is to evenly distribute tasks to all available threads. This is achieved through a "thread task manager" which ensures that all enqueued computations are assigned to threads. The thread task manager also assigns threads to computations in such a way that no thread idles while there are computations awaiting the controller's services. According to arguments and evidence provided in the paper [31], the task manager module makes it feasible for the controller software to evenly distribute work to all worker threads.

Although the intended objective of evenly distributing tasks to available worker threads

and improving the scale of the network is accomplished through this design, the worker threads are chosen based on the number of available processor cores in the machine and *not* based on the traffic to be processed. This also shows that the traffic to be forwarded in the network does not play any role in the design of the controller plane in terms of resource allocation. The research presented in this report argues that with the traffic (or its indication through statistical means) to be processed in the picture, one can achieve better allocation of threads and better memory efficiency.

2.1.3 Beacon

Beacon is another Java-based multi-threaded SDN controller [32]. It was developed to achieve three main design goals: An environment for easy SDN application development; provide the capability to start and stop applications at runtime; and also achieve high performance [32]. The notion of performance followed by the Beacon paper is that of the number of OpenFlow messages processed by the controller in a given period, or, put differently, the number of computations achieved by the controller per unit of time. The use of multiple threads is utilized to accomplish this performance objective, that is, in order to process more tasks per unit time, more threads are used to perform the tasks in parallel.

Beacon has two multi-threading designs (the version presented in [32]): shared-queue and run-to-completion. The shared queue designs has two stages: The first stage consists of threads which reads and assign OpenFlow messages to message types which can be interpreted by an *IOFMessageListener* system – a module or a set of steps able to execute the OpenFlow messages. The second stage is a set of threads which receive and process tasks from the shared queue in accordance to their type in the *IOMessageListener* pipeline. The run-to-completion method does not have the first set of threads, but each thread takes an OpenFlow message and run all its required computations.

This controller does not rely on any statistics of network traffic to chose the number of required threads or to chose an appropriate multi-threading design. The shared-queue or the run-to-completion could be selected based on workload demands. Therefore, this controller is also seen as one that does not consider the workload seen by the controller

plane for proper allocation of controller plane resources.

2.1.4 Floodlight

Floodlight [33] is an SDN controller project which provides an OpenFlow controller and a set of software applications to enable quick development of user-specific network application [34]. It is an open source Java-based multi-threaded controller developed by Big Switch [35]. The Beacon controller described earlier is reported to have formed the basis for the Floodlight project [32]. A clearly visible common goal between the Beacon and Floodlight projects is that of improving a developers productivity. Both these project provide platforms for quick development in popular programming languages such as Java, Python, C++, and others. The Floodlight architecture consists of a number of modules with which it implements the SDN controller functions. In order to illustrate the features of Floodlight which impact performance, three Floodlight modules are described: *FloodlightProvider*, *OFSwitchManager*, and *Packet Streamer*.

FloodlightProvider manages the controller's connections to switches and translates OpenFlow messages to events or signals which other Floodlight modules listen for [34]. The same module also controls the order in which these events are sent to the applicable modules [34]. This means that after the reception of OpenFlow messages by the *FloodLightProvider* module, there is a queue into which these messages are sent. This queue has a similar goal as the shared-queue described in the discussion of the Beacon controller. Therefore, this module is seen as implementing the shared-queue multi-threading design.

OFSwitchManager is responsible for managing all OpenFlow switches which are under the control of the Floodlight controller [34]. Threading and connections to switches are managed through the use of the Java's Netty library [34]. In this library, a switch event such as a `flow mod` is handled by a single thread from its start to end, that is, the thread executes the entire logic and computation required by the event [34]. This is similar to the functioning of the run-to-completion threading design suggested by the Beacon project.

The Floodlight project also provides a module for packet streaming called **Packet Streamer**, which can be used to stream packets exchanges between any OpenFlow

switch and the controller [34]. This is indeed a module of particular importance in this research report, because this module can be extended and used to study the traffic characteristics between switches and SDN controllers. However, there is no evidence which suggests that the *Packet Streamer* module is used in allocation of controller resources such as thread allocation. Hence, it can be stated that there is a need for one to explore the use of such a streaming service in the management of controller plane resources.

2.1.5 Trema

Trema is a programming framework for developing OpenFlow controllers by making use of Ruby and C as programming languages [36]. This is an open source project developed by NEC [37]. The goals of Trema include to support the entire process of developing SDN controllers and network application, and to encourage research in the field of SDN [36]. The Trema presentation document [38] highlights an easy development environment (IDEs and network emulation platforms); productivity achieved through the use of Ruby and C as programming languages and other features which makes it easier to develop applications in Trema as key attributes of the Trema project.

Although performance and scalability objectives are not discussed directly in [38], it is mentioned that Trema supports a multi-process model where applications (also called user modules) run in different processes. This is done to ensure that running processes do not affect each other negatively in an event of process failure. This means that in case a process crashes or fails, it does not affect other processes [38]. It is also mentioned that the user applications which implements network control tasks could run on different hosts (this was a planned module as of the writing of [38]), this can be considered as an effort to achieve a scalable controller plane. Due to the lack of evidence of use of traffic statistic in making controller plane decision in the available Trema documentation, it is argued that Trema does not have a technique to expose and use data plane traffic patterns to optimise the performance of the controller plane.

2.1.6 Summary on SDN Controllers

Evidently, these controllers (NOX, Maestro, Floodlight, Beacon and Trema) are counted among the widely used open source SDN controllers [29]. Several other SDN controllers are not described in this review, this is because the purpose of this review is to describe the general approaches followed in designs of SDN controllers and also investigate methods currently employed to achieve a scalable controller plane.

The evidence obtained through the review of literature on SDN controllers reveals that there is still a need to explore the nature of traffic between the data-plane and controller plane in SDN-based networks. This is because existing controller designs acknowledges the need for controller plane scalability, but these design do not indicate the nature of the load that the controller plane is required to handle. The work proposed in this research report, therefore, proposes to use statistical models to study the information contained in the traffic observed by the controller plane. It is believed that this information can be used to inform the designs of more effective controller planes.

2.2 Distributed and Hierarchical Controller Planes

Distributed controller planes are composed of two or more SDN controllers which are distributed at different locations of the network. The controllers keep a similar state of the network through a controller-to-controller synchronisation system. Distributed controller platforms discussed in this section are *Onix*, *Onos*, *OpenDayLight* and *Hyperflow*. A hierarchical controller plane discussed is *Kandoo*

2.2.1 A Distributed Control Platform: Onix

The Onix project is one of the first SDN controller plane initiatives that produced a platform on top of which a controller plane can be implemented as a distributed system [39]. Onix realizes that a controller plane for large-scale production networks needs to solve the following networking challenges [39]:

1. *Scalability*: As networks of today continue to grow in size and in the amount of carried traffic, the control software or controller plane is not expected to present

any scaling limitation [39].

2. *Reliability*: Network resilience or ability to handle equipment failure (on any interruption) is a critical attribute in networking. Therefore, the controller plane is expected to handle network faults successfully without compromising the functioning of the network.

3. *Simplicity*: The controller plane should provide easy to use APIs which enables for quick and simple development of network applications [39].

4. *Performance*: The computation of control tasks in the controller plane must be fast enough in order to avoid incurring increased packet delays in the network.

5. *Generality*: The control platform should provide APIs which enable all desired network applications to be developed.

The *Onix* controller platform consists of several physical servers which are used to run one or more Onix instances [39]. An Onix instance is a representation of a network entity (e.g. a port of a switch) as a high level abstraction on top of which a network controller application can be executed. An Onix API is used to expose network resources to the control logic implemented by control and management applications [39].

Another main component of the Onix controller framework is the *Network Information Base* (NIB). This is a data structure that produces and hosts a copy of the network-wide view or the state of the network elements (i.e. switches, topology, ports, etc.). The distribution of the NIB information to all Onix instances is another critical task which the NIB performs. In order to maintain a consistent view of the network across all the Onix instances, the Onix platforms relies on application-based logic [39] and not on the NIB data structure.

The scalability of the controller plane relies on the ability of the controller to handle all the NIB information without saturating system memory or degrading the controller plane's performance [39]. This means that as the number of devices or data plane resources represented in the NIB increases, the controller plane may run out of memory in a single Onix instance and end up degrading the performance of the system (or in the worst case, result in crashing the Onix instance or server). Three methods are

proposed by Onix to combat this scalability problem [39]: First, *partitioning* – the addition of controller instances such that a single instance is given control to a smaller number of data plane elements. Second *aggregation* – this is a method where an Onix instance aggregates the elements presented in its NIB and present them as an aggregated data representation to another Onix instance. Aggregation enables a number of nodes to appear as a single node to other controller instances [39]. Third, *consistency and durability* in the NIB information – this function is left for implementation by application developers. Each application should determine the consistency it desires from the NIB and dictate the level of coherency required.

This means that in case network traffic statistics such as link utilization are kept in the NIB, then applications can be developed to use this traffic statistic to control or manage the network. However, there is no evidence to show that such applications have been developed to manipulate the allocation of resources such as number of Onix instances or number of servers required in the controller plane. More importantly, network traffic statistics are not used to determine if an additional instance may be required or if the network connected devices might exhaust existing controller plane resources.

2.2.2 Open Network Operating System: Onos

Onos (Open Network Operating System) [24] is an SDN controller platform that follows a similar design approach as Onix. Onos provides a platforms on top of which SDN control planes can be implemented as distributed systems [24]. Unlike Onix which remained closed, Onos is an open source platform which uses open source technologies as its building blocks. The design objectives shared by these controllers include *scalability*, *high performance* and *availability* [24].

The key requirement of a distributed controller plane lies on its ability to keep a coherent network-wide view while ensuring that the distributed controllers are not overwhelmed by the amount of computations the data plane elements (switches, routers etc) demand. This entails that the network should be able to scale in terms of the number of data plane elements and the amount of information in the network-wide view data structure.

Onos confronts this challenge by employing an architecture composed of several servers. The number of servers in the system depends on the network's scale-out requirements. Each server runs the Onos control software and is given the mastership of controlling a subset of data plane elements [24]. Individual Onos instances are assigned the sole responsibility of exchanging information between a subset of switches which are under the control of a server (i.e. the server hosting the Onos instance of concern) and the global network-view system. This enables the system to keep a consistent network state among the servers which control different subsets of data plane elements.

The Onos paper [24] mentions that the first Onos design faced excessive computations between the network-wide information system and the controllers. Although the global view of the network was stored in the Apache Cassandra [40] data base system which is designed for high availability and scalability, the controller still suffered from poor performance [24]. Solutions to tackle this challenge are proposed in [24]. These include the reduction of read and write operations between the global network view and the controllers.

This problem is of particular importance in the research work presented in this report. This is because it reveals that the representation of network state (traffic, topology, port status, etc.) in the controller plane may result in adverse consequences if not tackled with the care it deserves. Therefore, this research proposes the use of statistical forecasting models which can reduce the communications between the controller plane and the data plane. These statistical models are intended to come up with methods to infer the network state with reasonable accuracy from observed traffic patterns. With forecasts that are reasonably accurate, then the need to continually poll the data plane for network state such as traffic statistics can be reduced.

2.2.3 HyperFlow

HyperFlow is an event-based distributed controller platform [41], which is also composed of distributed controllers running on servers. This system keeps a logically centralized controller plane through the use of a controller-to-controller synchronization mechanism called a *publish/subscribe* system. Similar to the Onos setup, the HyperFlow platform gives each controller the control of a subset of data plane elements.

Changes to the state of the data plane resources are communicated to the network-wide view through the publish/subscribe system. This means that the controller which observed or applied changes to their data plane elements publishes these changes as "events" which other controllers then replay to reconstruct the change in network-wide view [41].

The design choice of this controller as an event-based system relies on the events which makes changes to the network-wide view. Hence, this controller design appears to take into account the nature of controller-plane to data-plane (or in the opposite direction) traffic into consideration in the design of the publish/subscribe system. Nonetheless, the paper does not describe the traffic characteristics such as arrival rates of network events (e.g. `packet in` messages). For this reason there is still a need to explore such statistics in order to design a network-wide view system or a synchronization system which helps the design of more effective controller planes.

2.2.4 OpenDayLight

Another widely used open source distributed controller platform is the **OpenDayLight** (ODL) [20] controller plane system. ODL has recently announced the release of a new version of the controller platform called *OpenDayLight Beryllium* (ODL Be) in February 22, 2016 [20]. It appears as though the ODL Berryllium is the first release of the ODL platform which enables scalability through the use of distributed ODL instances. The ODL project also announced that the ODL Be controller instances can be configured to operate as a logically centralized controller plane [20]. The methods used to achieve a consistent network-wide view and scalability through distributed ODL instances are not discussed in the ODL documentation, perhaps this is because it is still an ongoing project. For this reason, the ODL controller is mentioned in this work because it is indeed an excellent initiative and is widely used in research around SDN and *not* to study its controller plane design approach like in the Onix and Onos discussion presented earlier.

2.2.5 Hierarchical Controller planes: Kandoo

The commonly known hierarchical controller platform is **Kandoo** [42]. This controller is designed from the basis that not all network events require access to a global network state. For example, a `packet in` event indicates the arrival of a new flow, and therefore, the global view of the network is required for path computation purposes. On the other hand, events like the polling a switch's port status may not require the global network state. For this reason, this controller setup advocates that some control tasks should be performed on a local processing device and others on a processing device with a global view of the network state. In other words, this controller perceives that some events should be processed in processing devices which are as close as possible to the data plane elements.

Instead of proposing that some control tasks be moved to the data plane (like in solutions proposed by DIFANE [43]; DevoFlow [44]), the Kandoo controller platform postulates an architecture with two levels: A root controller plane responsible for events that need a global state of the network and a second level controller plane responsible for local events. The local controllers are responsible for the control of a group of switches, but in case an event which requires the network-wide view occurs, this event is sent to the root controller for processing. In a similar manner, in case a root controller wants to modify a flow table of a switch, it delegates the task to a local controller closest to the switch of concern.

The deployment of a Kandoo-based network depends on a network's requirements. For a network that is composed of software switches, the local controller is preferably run from the same processing unit as the software switch. In a network that mainly consists of physical switches, the local controllers are given groups of switches to control. These controllers are kept as close as possible to the switches. In both scenarios, the root controllers do not have direct connections to the switches, so they access the state of the switches from the local controllers and control the state of the data plane through delegation of tasks to the local controllers. Similarly, local controllers do not have direct access to the global state of the network, but accesses it through the delegation of tasks to root controllers.

The setting of such networks is static and is not informed by traffic statistics. For

example, locating the local controller in the same processing unit as the software switch can have adverse consequences on the performance of both the local controller and the software switch. This is because with the controller and the switch relying on the same computing resources, more traffic switching on the soft switch could mean more resources given to the switch, and in a similar manner, more controller computations could also mean the dedication of more resources to the controller. As a result, this controller plane design also reveals the need for careful consideration of the network traffic before the deployment of controllers as local or root level controller devices.

2.2.6 Summary on Distributed Controller planes

The general designs of SDN controllers as well as the methods employed to realize high performance and scalable controller planes have now been discussed. These discussions reveal that one of the most pressing issues on controller platform designs is around the network-wide view which has to be kept up-to-date and coherent in a case of distributed controllers. The network-wide view has a direct impact on the scalability of the network being controlled by an SDN controller. This is because both the computational and storage capacity required to keep a usable (and up-to-date) network-wide view should be catered for by the controller platform.

In a situation where the controller platform has limited resources to handle the network-wide requirements, the data-plane resources represented in the network-wide view may need to be limited or reduced. The network's scalability is directly affected by the limitation in controller plane resources. According to the observed literature, there are only two ways through which one can keep the network-wide view requirements reasonable on storage and computing power: Firstly, reduce the amount of information representing data-plane resources. Secondly, decrease the number of devices under the control of a given controller plane. Both scenarios talk to reduction in network's scale, the first reduces the amount of resources which can be controlled from a remote controller and the second reduces the number of network connected devices.

It has been pointed out that the statistical properties of the traffic between the controller plane and the data plane have not been extensively studied and applied in designing controller planes. It is believed that some elements of the controller plane

could greatly benefit from such statistics. The research presented in this report proposes to employ statistical forecasting models to forecast the traffic of concern. In case the traffic is found to be predictable with reasonable accuracy, the forecasts could be used to represent the state of data plane resources. This could reduce the number of interactions between the controller and the network-wide view database system and therefore prevent controller congestion.

2.3 SDN Controller Performance Measurement Tool

The performance of SDN controllers is judged through measurements of the controller's *throughput* and *latency*. The controller's throughput measures the number of flow initiations or path requests an SDN controller handles in a given period (i.e. per second). The controller's latency measures the controller's response time, that is, the time it takes for the controller to respond to a data plane element's request. A widely used open source tool for controller benchmarking is Cbench [45], which was developed in 2012.

Cbench is used to emulate data plane elements of an SDN-based network such as OpenFlow compliant switches. The emulated network of data plane elements is connected to the SDN controller to be evaluated. Cbench operates in two modes: latency and throughput mode [45]. In latency mode, each data plane element is configured to send one request to the controller, and after sending the request, the data plane elements waits for the controller's response before sending the next request [45]. Hence, the latency mode is used to measure the processing time of the controller in conditions of few flow path requests [45]. In throughput mode, the data plane elements send as many requests as possible to the controller while Cbench monitors the controller's processing time. For this reason, the throughput mode measures the maximum rate at which the controller plane can respond to data plane requests [45].

Cbench is indeed a useful SDN controller benchmarking tool, but it does not take into account the workload distributions the controller is likely to face in real life deployment. This means that the load which is sent by the Cbench data plane elements does not follow traffic patterns which have been observed in real life networks such as enterprise WANs, LANs, data centers etc. In the SDN literature observed at the time of the

writing of this report, no research work was found to have focused on the nature of the traffic patterns and workload distributions which are seen or likely to be experienced by SDN controllers in real life deployments.

It is believed that the inclusion of realistic traffic patterns in tools like Cbench will improve the benchmarking of SDN controllers. For example, with the traffic patterns known, it will be possible to evaluate SDN controllers to be deployed in data centers with workloads or traffic patterns which are likely to be encountered in data centers. This is one of the reasons the work presented in this report aims at studying the traffic patterns between the SDN contoller planes and data planes.

2.4 Traffic Analysis in SDN

The utilisation of network resources (e.g. switch port usage, link utilization, etc.) in a traffic carrying network is assessed through the use of network traffic monitoring systems. A traffic monitoring system captures, processes and presents usable traffic statistics to a network management or control application [46]. The statistical information made available is used to perform such tasks as traffic engineering; load balancing; enforcement of SLAs (service level agreements); billing and others [22]. Thus, the importance of these tasks require that network management or control applications be furnished with accurate and timely network statistics [22].

Although the accuracy and timely requirement of the network resource information is essential, it is still required that the monitoring systems do not incur significant network overhead. Traditional network monitoring systems are designed for specific cases; use specialized instrumentation and incur significant network overhead [22] [46]. Therefore, the primary goal of the design of network monitoring systems is to provide accurate and timely information without incurring significant network overhead [22].

The monitoring methods used in Traditional IP systems are classifiable into to two classes: direct and sampling-based monitoring [46]. In direct methods, the resource of concern is continually monitored. Hence, these methods are known to be accurate, but suffer from high network overhead. In sampling-based monitoring, the resource whose utilisation is in question is monitored at specified time intervals. These methods incur less network overhead compared to the direct methods. The less network overhead in

sampling-based monitoring methods is achieved at the expense of accuracy.

The objective of the review of network resource analysis and monitoring tools in this report is to determine if direct and sampling methods are applied (or applicable) to aid the designs of controller plane platforms in SDN networks. The network measurement and monitoring tools currently used in SDN are taken from traditional IP or Ethernet networks [22] [46]. Therefore, the monitoring frameworks from IP networks are discussed first (NetFlow and Planck) and then those designed specifically for SDN follow (Payless and OpenTM).

2.4.1 NetFlow

Network Monitoring systems consist of the following components: Firstly, a method to capture the statistics from a network element (switch, router, etc.). Secondly, a statistics processing unit, normally called a collector. Thirdly, an API or a technique which is used to expose the statistics to a management or control application [22]. NetFlow is a network monitoring system developed by Cisco [47], it consists of the following components [48]:

- An instrumentation system to collect statistics is coupled in Cisco devices (routers, switches, etc.).
- A collector (or storage) to which the information collected from the devices is sent.
- A graphical user interface or an API by which the management or control applications access the resource utilisation statistics.

For sampling accuracy, the tool uses direct measurement techniques, and to reduce measurement overhead, the sampling-based method is also applied.

2.4.2 Planck

Planck is a monitoring tool designed for use in SDN based networks [49]. Its main design goal is to provide a monitoring tool with the capability to extract network

information in order of milliseconds while maintaining less network overheard. Planck utilizes oversubscribed port mirroring to achieve the extraction of network information at 280μs – 7ms [49]. One notable feature of this tool in SDN is the capability to expose the statistics to an SDN controller. The controller can listen for events from the collector or subscribe to them.

Although the implementation of this tool may inform the SDN controller of network events timely, it can not be guaranteed that the SDN controller will have enough resources to react to the event at the time the event is reported. This is because the tool does not have a method to tell if the controller has enough resources to compute a given tasks within a required time frame. It is important to point out that no matter how fast the events are reported to the controller, it is still the controller's computational power which determines the time within which it can react to a network event. For this reason, the research presented in this report proposes a method which can give network application an indication of controller plane resources usage.

The second drawback that can also be pointed out in the design and usage of Planck is the use of port mirroring for statistics collection. Although this method is effective and accurate, port mirroring is an attribute of switches, which must exist in a switch before Planck can be applicable to it. The reason for this to be considered a drawback is that the mirroring capability may not exist in some switches, and since switch designs are proprietary, it is not known if the mirroring capability of different switches will provide same results when used by Planck.

2.4.3 Payless

Payless is a low-cost SDN traffic measurement and monitoring tool that operates on top of the northbound API of an SDN controller platform. Like all network monitoring systems, Payless' objective is to obtain accurate measurements of network resources while keeping the overhead of the measurement system low. In efforts to ensure accuracy of the measurements, Payless employs adaptive statistics collection algorithms which ensures high accuracy and less overhead [46]. These are algorithms which vary the frequency of the polling of information from the network elements. These algorithms have been found to be effective in many applications where polling of information is

required [46].

The design of Payless assumes that the implementation of a monitoring system on the SDN controller might make controller designs more expensive and complicated, so the approach taken was to create a separate layer on top of the northbound API [46]. It is this layer that implements the Payless system. Control and management applications access the statistics made available by Payless system through a RESTful API which is also provided by the Payless system. The authors of the Payless paper in [46], indicate that Payless achieves the intended measurement accuracy and has less network overhead. This system also concentrates on data-plane resources and not controller plane ones. Therefore, although Payless may satisfy monitoring the data plane, there is still a requirement to monitor the usage of the controller plane resources.

2.4.4 OpenTM

OpenTM is an OpenFlow based traffic estimation system [50]. This system was designed under the realization that standard traffic estimation tools rely on statistical models for inference of volumes of traffic between hosts in a network [50]. Consequently, OpenTM uses the direct measurement approach which employs OpenFlow's statistics gathering mechanisms. The traffic matrix estimated by OpenTM is that of traffic volumes between origin-destination (OD) pairs. Since the OpenTM traces the flows in the switches to generate the matrix, it is required to periodically poll the switches. The periodical polling of switches has been reported to be one of the causes for high network overhead in such systems [22]. Hence, one disadvantage of this system might be high network overhead. Moreover, this tool does not provide any information on controller plane loading.

2.4.5 Summary on Traffic Monitoring Tools

The literature on SDN-based network monitoring contains several other traffic measurement and monitoring systems which are not discussed in this work. These include MicroTE [51], FlowSense [52], OpenSketch [53], OpenSample [54], and others. All of these systems basically concentrate on monitoring the network's data plane and *not* the controller plane load. There is no indication that the systems discussed have been

applied to monitor the amount of tasks performed by the control plane, which is the prime concern of this research work.

2.5 SDN Traffic Engineering Schemes And Other Solutions

Networks traffic engineering is the analysis and regulation of the traffic carried in the network's data path with an objective to optimize the performance of the network [22]. Factors which are normally of great concern in network performance optimization include load balancing; minimization of power consumption; reduction of network overhead from monitoring systems; and others [1]. These tasks are performed through modifications of the carried data-plane traffic, which in SDN is a task of the controller plane. Hence, these applications are normally implemented on top of the controller plane as management or control applications. Traffic engineering applications are reviewed in this work to determine the feasibility of their use in controller plane load balancing initiatives.

2.5.1 Aster*x

Aster*x [55] is an SDN-based load balancing application which is used for the balancing of web traffic over WANs and other unstructured networks such as enterprise networks [55]. The Aster*x project observes that it is a common approach for web sites to balance load over multiple HTTP servers [55]. The challenge with this approach is that it does not consider the load of the HTTP servers [55]. This means that such approaches may result in web clients waiting longer periods of time for responses, because a server that receives a request while busy with other tasks may take a longer period of time to send back a responses to new requests compared to a server which receives requests from an idle state.

In efforts to reduce the average response time, the Aster*x project employs an OpenFlow-based SDN network to connect a set of distributed servers which hosts web services. The servers connected in the network are given the same IP alias, so that when a request arrives, an SDN controller decides the server to which the request will be sent as well as the path to be used. The system consists of three key modules: Firstly, *Flow Manager* is used to manage the flows of traffic to the servers as decided by the

applicable load balancing algorithm. Secondly, *Net Manager* is responsible for keeping the network-wide view of the network. Lastly, *Host Manager* is used to monitor the state of the servers and the loading they experience. Armed with these modules and the routing algorithms, the system enables operators to balance load successfully and increase the capacity of the network by simply adding network resources such as servers and switches to the network arbitrarily [55].

In view of an SDN controller plane as a system which is expected to provide responses to network events within a very short period of time, the Aster*x approach to balance loading on servers by taking into account the current load on the servers can be applied to controller planes. For example, in a distributed controller plane, the controllers operate in a similar manner as servers which provide services to their clients (e.g. web clients), so the current tasks or received requests to the controller plane need to be taken into account before assigning tasks to the controller. The research question to be answered in this report aims at studying the feasibility of using the traffic in the southbound interface of an SDN network to determine the type of traffic patterns seen by the controller plane and also attempts to forecast this patterns. It is believed that answers to the questions (i.e. the research question) will provide a starting point to the application of such methods as Aster*x to the SDN controller planes.

2.5.2 SDN Controller Placement

In [56], an SDN controller placement problem is introduced. This problem aims to open a discussion on the number controllers required in a network and their locations with respect to the data plane elements [56]. It is also aimed at providing answers to concerns raised around the controller plane response time; the scalability challenge and availability [56]. The study, therefore answers two key questions [56]:

"How many controllers are needed? and Where in the topology should they go?"

According to the study conducted in [56], the number of required controllers depend on the desired controller plane response time; the network topology and the performance metric (containing network parameters such an average-case latency). This means that traffic carried in the data plane and the communications between the controller plane and the data plane can be used to design controller planes. It was also discovered (in

[56]) that there are no general rules that can be used to place controllers. Instead, controller plane designers and network operators are given guidelines to follow when adding a new controller node. The research proposed in this report explores the role that can be played by data-plane to controller-plane traffic statistics in determining the required controller plane resources. Therefore, it can be viewed as a continuation of the work started through the controller placement problem given in [56].

2.5.3 Scotch

Scotch [57] is a mechanism that uses an Open vSwitch (OVS) based overlay network to elastically scale up the data-plane to controller-plane control channel. The Scotch design stems out from the observation that the control channel between a switch and an SDN controller may become a bottleneck in an event of a large number of reactive flows, because each reactive flow requires a controller plane's services [57]. A normal switch's data plane forwarding capacity is larger than that of its control path by several orders of magnitude [57]. Therefore, Scotch exploits the availability of high capacity data-plane forwarding of vSwitches to elastically scale up the control path of a switch.

The Scotch based network is composed of the SDN data plane physical switches which are connected to an overlay network of vSwitches via tunnels. Both the vSwitches and the physical switches have control paths to SDN controllers; and the overlay network is also connected to the end hosts. The system operates as follows: When the control path of the physical switch is overloaded, new packets which require access to the controller are forwarded to the vSwitch via the high capacity forwarding facility of the physical switches, and the vSwitch which received the incoming packet then contacts the controller via its high throughput control channel and ultimately forwards the flow of packets to its destination [57]. This technique does not result in increased packet delays, because a large number of vSwitches are used in the overlay network compared to that of physical switches and the vSwitches have high throughput control channels as they (vSwitches) run on powerful CPUs.

The Scotch design achieves an elastically scalable control path, but it does not take into account the processing power of the controller plane. This means that a high capacity control channel can be achieved, but, if the controller plane does not have

enough processing power, the controller plane may still become a bottleneck. This highlights the need that the control channel traffic as well as the controller plane's processing power should be taken into consideration when addressing the controller plane scalability issue.

2.5.4 ElastiCon

ElastiCon [58] is an elastic distributed controller plane. This controller architecture realizes the need to consider prevailing data-plane to controller-plane traffic load to assign distributed controllers to switches or other data-plane resources (the paper [58] considers switches only). The idea behind ElastiCon is to set a controller plane load threshold which when exceeded, the number of controllers in a pool of controllers is increased or some switches are migrated from a highly loaded controller to a lightly loaded one. A protocol to facilitate the switch migration is developed as well as a technique to decide if switch migration is required or if the number of controllers need an increase or decrease [58].

A load estimation module uses the average message arrival rate to determine the loading on controller's CPUs - the authors determined that the message arrival rate observed by the controller plane is proportional to CPU load [58]. The ElastiCon project can be directly comparable to the work proposed in this research report, because the central point of the two research projects is to consider the traffic between the data and the control planes of an SDN network to inform the required number of controller plane resources. One aspect of the data-plane to controller-plane traffic that is not presented in the ElastiCon project is the traffic pattern or characteristic. This means that it is not possible to tell if the traffic is from a WAN, a Data Center or an Enterprise LAN or if the packets used to load the controller plane are generated by a statistical distribution. The feasibility of forecasting the controller plane workload is not considered in the Elasticon project. This is another attribute of the traffic properties which is necessary to avert controller plane congestion. Thus, the work proposed in this report considers the lack of controller traffic characterization and forecasting attempts as a "gap in knowledge" and attempts to fully explore the traffic and determine the feasibility of forecasting it by using statistical modelling. It is also believed that armed with the full traffic characteristics, one can improve projects such as ElastiCon. For example,

knowing the applicable traffic distribution, one can generate realistic controller plane workload for testing purposes.

2.6 Chapter Summary

The primary goal of the research presented in this research report is to study and analyse the traffic patterns of the communications which take place between the data plane and controller plane devices in SDN-based networks.The outcome of the research will reveal if it is possible to use these traffic statistics to design effective controller planes which will be scalable to meet today's network needs. Therefore, the review of literature presented in this chapter concentrates on controller plane solutions which have been devised to combat the scalability challenge.

SDN controller planes are discussed with emphasis on methods used to alleviate the controller congestion. It was found that most controllers relied on the addition of processing devices (servers) or on multi-threaded controllers to increase the controller plane's capacity to accommodate larger data plane resources. These solutions are indeed effective to some extent, but the challenge observed is that the solutions do not rely or make use of the traffic (or load) seen by the controller plane to inform their design approaches (multi-threaded or increased servers). Therefore, there is still a need to explore the controller plane workload extensively in order to determine the feasibility of using it to inform controller plane solutions.

Distributed and hierarchical controllers also emerged as solutions to confront the scalability challenges. These are effective and one can even state that these designs are the ultimate solution to the scalability challenge in SDN-based networks when looking at the state of the art. However, it has been observed that although these designs acknowledge the need for a distributed system, the method to determine the number of distributed controllers or their locations in the network are not based on the envisaged controller plane workload. In fact, there are no methods used to forecast the controller plane workload. Thus, recognizing the need to forecast the controller plane workload, the research proposed in this report aims at employing statistical forecasting models to forecast the controller plane workloads.

Traffic engineering and traffic analysis methods in SDN networks were also reviewed. It

was found that these methods concentrate on the management and control of data plane elements. Moreover, some of these methods, the monitoring tools in particular, require special instruments and are normally designed for specific tasks. Another drawback in monitoring systems is the network overhead that such systems normally impose on the network. Traffic engineering approaches also concentrate at solving data-plane related problems, although they are implemented on top of the controller planes. ElastiCon and Scotch are the only traffic engineering approaches which were found to be taking the controller workload into account in the designs of their controller plane solutions. These solutions also lacked thorough analysis of the data-plane to controller-plane traffic statistics. As a result, at the core of the research presented on this report is the desire to fully understand the controller plane workload, and attempt to forecast it.

3 NETWORK PROTOTYPE

Network research activities such as development of new network protocols (or evaluation of existing ones); evaluation of network performance (e.g. QoS); network traffic analysis and others, require the network researcher to conduct credible experiments. The requirements of the experiments are largely determined by the research in question. However, it can be pointed out that the elementary requirements of experiments performed in prototypes of network systems include network topology – networking devices such as servers and links; network traffic – user or application generated data; and protocols – rules used for the exchange of information among networking devices. In these prototypes, network elements may be the real hardware devices or their software representation through the use of network simulators or emulators.

The digitization of today's societies has made computer communication networks a critical infrastructure requirement for many governments around the world. Some countries consider access to communications as a basic human right. For example, South Africa's National Broadband Plan (NBP) aims at extending broadband access to rural areas, which forms part of the government's national development plan. Due to the extensive usage and importance of live networks in today's societies, it is impossible to perform research experiments on existing production networks [59]. Therefore, the network research community relies on three platforms for experimentation: *network simulators*, *testbeds* and *network emulators* [59]. These networking research platforms are expected to be cost effective (in terms of resources and time) and credible for the research results to be considered valid and applicable in real life networks.

Network simulators employ the concept of modelling where the behaviour of a real-life system is simplified to suit the situation of concern. In network simulations, mathematical models are used to generate the network traffic; model the behaviour of queuing systems in switch buffers; represent physical channels (e.g. the wireless channel); and other network functions. Due to the simplifications which go into these models, the

lack of realism and practicality is a major concern with network simulators [60] [61] [59].

A study on the reliability of results obtained from network simulators is presented in [62]. This study found that results obtained from different network simulators are not consistent due to lack of accurate modelling of physical quantities such as the wireless channel in wireless networks or other quantities in the media access control (MAC) layer. The Network simulator NS-2/NS-3 [63]; OPNet [64] and QualNet [65] are examples of network simulators which have been extensively used in the networking research community [59] [60] [62].

A research on traffic statistics, like the one presented on this report, requires that the network prototype be as realistic as possible. Realism is considered important in this research work. This is because the research results are aimed at the characterisation and forecasting of the traffic, which in case the research experiment is not realistic will give invalid results. In light of the required realistic conditions, network simulators are not preferred in this study.

Testbeds for networking systems are facilities which host servers, switches and other network elements. These resources can be hosted in a single location or spread in various locations [61] [59]. Unlike network simulators, testbeds host the real hardware which make up the networking devices. For this reason, results obtained from testbeds are considered more valid or trusted when compared to those from simulators. The resources hosted in the testbed facilities are finite, which means that in some instances users may have to queue for resources [59]. Therefore, it is not known if a testbed user will always have all the required networking resources for an experiment or if given resources will scale to meet all the experiment's network requirements. This means that the user does not have full control of the network resources provided by a testbed. It is due to these reasons that network testbeds are not considered the best option for the research work under discussion. Examples of networking testbeds include Emulab an initiative of the University of Utah [66]; Global Environment for Networking Innovations (GENI) which is a large communications infrastructure spread across several states in the US [67]; OFELIA an OpenFlow project based in Europe [68] and others.

Network emulators on the other hand, emulates all the aspects of a real network through software running on a computer [60] [59]. The software captures the representation or virtualization of the network elements (switches, routers, servers and packets); performs the configuration of the network and run real network tasks such as sending of packets between hosts and running of applications on the virtual servers [59]. The code and configurations used in the network emulator are the same as those which runs on real hardware. Both network emulators and networking testbeds produces more reliable results compared to network simulators [59] [61]. The SDN community largely depends on Mininet [23] (explained in Section 3.2) for quick SDN network prototypes. Mininet was developed by the computer networking group at Stanford University [59] and has become a key enabler for testing SDN networking ideas such as new SDN controllers, new OpenFlow compliant switches and other features of SDN-based networks.

The investigation conducted in the research work presented in this report aims at revealing the behaviour of the traffic between an SDN network's data plane elements and controller plane. The two possible forwarding behaviours of an SDN network are discussed in the problem statement of this research (Section 1.4.2), that is, reactive and proactive forwarding. In both these forwarding mechanisms, the controller plane is tasked with exclusive control of the forwarding tables of switches, routers and other data plane elements. This means that the controller-plane's workload comes as a result of traffic flowing in the data plane. Therefore, the requirements of the network experiment are as follows:

1. Network topology - WAN, LAN or Data Center.
2. Traffic generator - traffic generation should be realistic and take into account the topology.
3. Data plane elements such as switches or routers - arranged in accordance with the selected topology.
4. SDN controller plane (single instance or distributed).
5. Data capturing instrument in the control channel.
6. A statistics processing tool such as MATLAB or GNU R.

It is also perceived that the networking research community has not explored time series forecasting models extensively in recent years. The reluctance to testing time series techniques in networking can be attributed to the shortage of realistic network prototype tools which network researchers suffered in the past 10-20 years. As an example, network simulators (e.g. NS-2) which dominated the networking in research area in the past relied on mathematical models for traffic generation. Due to the lack of realism in these tools, time series forecasting methods could not be explored fully. In light of the emergence of realistic network emulators such as Mininet, testbeds such as Emulab and others, the work presented in this report aims at studying the applicability of time series methods in forecasting of an SDN-based network controller plane workload.

This chapter is structured as follows, Section 3.1 discusses the technologies which enables network emulation through tools such as Mininet. Section 3.2 describes Mininet with a particular focus on the features which makes it applicable to the work under question. Section 3.3 discuses data centers and the network topology on which the ideas presented in this work will be tested. Sections 3.4 presents a software tool that was designed to replay network traffic on Mininet. A chapter summary is given in Section 3.5.

3.1 Virtualization in Network Emulators

Network emulators run on a single computing hardware such as that of a desktop computer, laptop or even a raspberry pi (Mininet). The network components emulated in the network emulator share the resources of the core hardware through virtualization, a concept of sharing computing facilities or resources of a single expensive hardware that was introduced by IBM in the 1960s [7] [69]. It is generally accepted to define virtualization as the sharing of computing resources through a software abstraction layer called a Virtual Machine Monitor (VMM) or a hypervisor [7] [69] [70]. This enables one or more full computer Operating Systems (OS) to run on top of the hypervisor or VMM. It is through the VMM that the guest OS accesses the abstracted resources of the computer's hadware [7]. Figure 3.1 [7] [69] [70] depicts the architecture of a computing facility that runs hypervisor-based virtualisation.

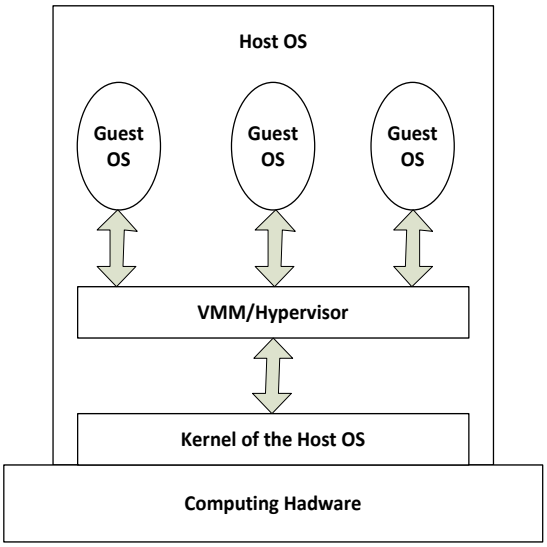


Figure 3.1: Hypervisor-Based Virtualization – Adapted From [7].

Two categories of virtualization can be pointed out: hypervisor-based virtualization and operating system level virtualization [59]. The former is also called full virtualization, because it enables the emulation of hardware where a full operating system can run independent of the host's operating system (Figure 3.1). The latter is lighter compared to full virtualization, it enables multiple user space instances of the same OS to run on top of the same kernel [7] (shown in Figure 3.2) . This is also called container-based virtualization [59] [7]. These containers or OS level virtualization isolates processes and resources in the user space, which in turn allows for multiple processes to run in parallel and independent of each other.

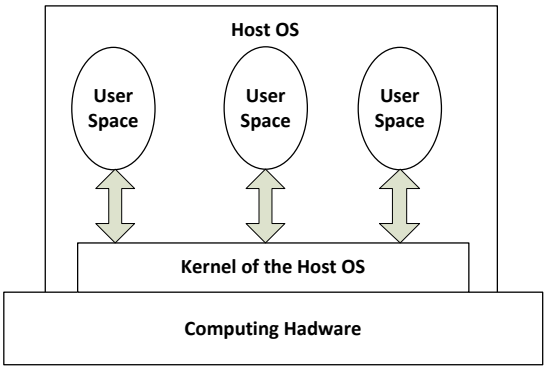


Figure 3.2: OS-level Virtualization – Adapted From [7].

In Figure 3.1 the VMM or hypervisor layer presents abstracted features of the hardware

to the guest OS whereas in Figure 3.2 [7] [69] [70] the virtualization layer provides abstracted features of the host's operating system to multiple user space instances. Due to the emulation of hardware in full virtualization, network emulator which employs this type of virtualization face high performance overhead [70]. Container Based Emulation (CBE) does not face the challenge of excessive performance overhead, because it does not emulate hardware. In view of performance and scalability benefits offered by CBE, Mininet, a container based network emulator was selected for the study being undertaken by this research project.

3.2 Container-Based Network Emulator: Mininet

Mininet is a container based network emulator which creates an environment where emulated hosts, switches and links run real network applications on a single OS kernel [59] [61]. These software-emulated network elements also use real kernel code which can be used unchanged on real hardware [59]. Mininet benefits from the latest developments in Linux OS-level virtualization: The concept of Linux *containers* (LXC), *network namespcaes* and *Virtual Ethernet* (veth) links. These features are used to emulate hosts, switches and network links. A brief description of the *LXC*; *network namespaces*; *veth* and their use in Mininet is given below:

Linux Containers (LXC) enables groups of processes to be isolated and run independent of each other [71]. The container's task is to host the groups of processes running on the same OS kernel and give them an independent view of the system's resources such as the file system; process IDs, network devices and user names [59]. This is lighter than in the concept of hypervisor-based virtualization which emulates hardware and run separate full operating systems. LXCs employ Linux features called namespaces and control grouping to realize the isolation of processes [71]. There are six namespaces defined in Linux [71], but, the one which finds great applicability in Mininet is the *Network namspace*.

Network Namespaces allows for the allocation of the OS's networking devices to isolated groups of process such that the networking devices operate independent of each other [71]. As an example, a network interface allocated to a group of processes in a namespace will have its own ARP caches and routing tables which other interfaces

(in other namespaces) cannot interfere with [61]. It is for this reason that processes in different network namespaces can run similar networking services on the same port (e.g. a web server on a port) or IP without interfering with each other [59].

Virtual Ethernet Links are created by virtual Ethernet pair, also called veth pairs. The veth pairs connect two network interfaces just like a wire would connect two points or a pipe connecting two points [61] [59]. The interfaces realized through the veth pair are seen as real Ethernet ports to the system and can be used to connect to virtual switches [59].

These linux OS-level developments have enabled Mininet to emulate the following network components:

Virtual hosts are network namespaces which contain the network state [59] [61]. It is through these namespaces that the virtual hosts are realised. Groups of processes running in these namespaces are provided with ownership of networking devices and all their features and functions (interfaces, ports, IP, routing tables etc.) [59]. For this reason, Mininet defines a virtual host as a shell process contained in its own network namespace [59] [60] [61].

Links are created by a veth pair. The veth pair connects hosts (i.e. namespaces) and switches.

OpenFlow Switches used in Mininet are the Open vSwitches (OVS) [72]. The OVS switches have the same packet delivery capabilities as a real hardware ones. These switches are also OpenFlow compliant and can be controlled by an SDN controller.

SDN Controllers, Mininet comes with its own openflow controller which can be attached to the network and can also be connected to a remote controller through an IP-level connection. In the work under discussion, only remote software controllers are used.

Due to lack of performance fidelity in the original Mininet version, the Mininet Hi-Fi (High Fidelity) version was developed [59] [60] [61]. Mininet Hi-Fi, relies on a concept called performance isolation, where resources used by the hosts, switches and links are isolated so that one process does not consume all available resources [59]. Through the development of Mininet Hi-Fi, network research papers produced by real hardware

and networking testbeds can be reproduced with reasonable accuracy [61]. The success of Mininet Hi-Fi in reproducing research that was conducted in real hardware proves its applicability in the work proposed in this report and in other network prototypes. With Mininet selected as the network emulator to be used in this research, the next section describes the network environment (a data center) to be explored.

3.3 Data Centers

Data centers are clusters of computers/servers which are dedicated for the provision of computing resources - exabytes of storage facilities and petaflops of computing power [73] [27]. A multitude of software applications are hosted and run from data centers, these include Web-based services (e.g. Google, Bing, Yahoo!, Twitter, LinkedIn etc.); cloud computing services (e.g. Amazon Web Services (AWS), Dropbox etc.); big data applications (e.g. Hadoop, Spark etc.) and scientific research computing tasks. Enterprises, universities, corporations and many other institutions rely on data centers for computing resources and services. Therefore, it is without a doubt that data centers are the infrastructure of critical importance in computer networking.

The design of the networks which connect a data-center's networking devices *inside* the data center faces a larger number of engineering constraints compared to the designs of WANs and other globally spread networks like the Internet [8]. For instance, a data center network interconnects a large number of similar devices in a relatively smaller physical space compared to the Internet which connects a wide-range of devices which are spread around the world [8]. Moreover, the data center nodes are interdependent [8], and therefore require robust interconnections to each other. These requirements inform the design of both the physical connections (optical fiber or copper cables) as well as the network protocols. Examples of network architectures designed to suit the data center's nodes include Clos-based networks such as the fat-tree [8] [74] [75]. Routing strategies which are considered of particular importance to data center networks are multi-path routing strategies like ECMP - equal cost multi-path routing [75].

Although a lot of work has been done in the design of the data center networks and the applicable routing protocols, many studies (including [26], [27], [28], [8], [74]) still reveal that data center traffic patterns, that is, the workload seen by the data center's

computers/servers and links is not well understood. The traffic patterns presented in these studies differ significantly due to a number of reasons, one of them, the applications running on the different data centers on which the investigation were performed are different and present different computational requirements on the network. Furthermore, data center traffic traces are not publicly available due to security reasons.

The wide-range of applications running in a data center and the different computational requirements these applications demand on the data center network is seen as the most attractive feature for the use of a data center as a case study in the research presented in this report. This is because in an SDN-based network, the diversity of the traffic flows (in terms of source and destinations) require flow table entries which will cater for all the traffic flows. In case a table miss event is encountered due to the absence of a flow table entry, then the controller plane need to be contacted. The data center is therefore taken as the network environment which can expose the SDN controller to a wide range of data plane requirements. Moreover, many studies have identified the need to employ a centralized controller plane in data centers, these include, Hedera [75], HULL [76], MicroTE [51] and Freeway [77]. It is due to the applicability of centralized controller platforms in improving data center routing strategies that SDN and centralized controller platforms will play a vital part in designs of data center networks.

3.3.1 Data Center Topology

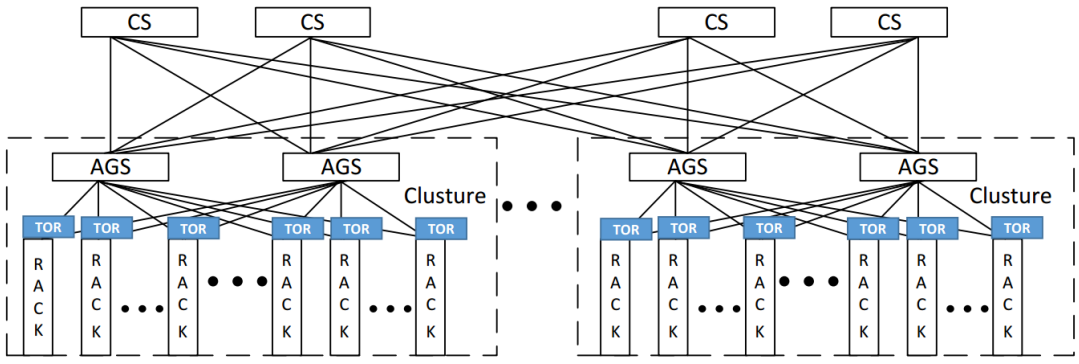


Figure 3.3: A 3-tier Data Center Network Topology – Adapted From [8].

.

The most dominant topology of data center networks is the 3-tier tree network depicted

in Figure 3.3. The servers/machines are placed in racks and connected to top of the rack (TOR) switches. The TOR switches connects the racks to aggregation switches (AGS), and the AGS are connected to core switches (CS). Most studies (including [75] and [73]) indicate that machines of the same rack are normally connected via 1Gbps links and are able to communicate at the full capacity of the links. Machines linked through the use of switches in different levels of the tree may suffer from limited bandwidth [73]. In efforts to make up for the possible bandwidth limitations, links between TOR and AGS are normally 10Gbps, and the same capacity is used for the links between the AGC and the CS [8].

3.3.2 Data Center Traffic Generation

It has been mentioned earlier that the research presented in this report aims at revealing the traffic patterns observed by the controller plane of an SDN network, and also determine the feasibility of forecasting the workload of the controller plane. In SDN networks which use reactive traffic forwarding, before asynchronous messages can be sent to the controller plane, there has to be traffic in the data path of the network which initiates these messages. This means that for realistic traffic to be observed by the controller plane, the traffic carried in the data path of the network has to be realistic. Consequently, the validity of the results obtained from this report also depends on the method of traffic generation – its ability to produce realistic traffic. In view of this requirement, the following rules are set for the traffic generation methodology.

1. **Data-plane traffic diversity:** The traffic should be generated on a flow basis (traffic between origin-destination pairs) and the flows should be diverse in order to request different path computations from the controller. This is an important requirement, because it is the arrival of new flows that results in the controller plane workload (for routing instructions) and *not* the existence of a known flow in the data plane elements. Therefore, diverse flows will ensure that the reactive controller plane setting chosen for this research will encounter a reasonable amount of path computation tasks – this is considered the controller workload.

2. **Topology-aware traffic generation:** Different network topologies encounter different traffic patterns and in most cases topologies are selected based on the

anticipated traffic. For example, the data center traffic characteristics study presented in [28] indicate that the majority of traffic in data centers running applications like Hadoop is kept in the same rack. In SDN-based networks, the topology is important because it is the main input to algorithms which determine the best route for a flow of packets. Hence, traffic generators need to take the topology into account.

3. **Reproducibility of result:** Research result need to be validated by other researchers in a given area of study. Therefore, the method of traffic generation should be publicly available in order to allow other researchers to continue with the research or verify the results.

4. **Traffic realism:** Real traffic traces are hard to find due to security reasons. For this reason, traffic generation which takes into account features of traffic which have been observed in real life is considered adequate in this research.

The *data-plane traffic diversity* requirement has an impact on both the arrival rate and duration of flows. Data center traffic studies presented in [27] and [28] present different views about the duration of flows in data center: In [27], it is reported that most flows last for less than 100ms, and in [28] it was found that most flows last for less than 10s, which is 100 times more than the duration of flows found in [27]. This difference in results can be attributed to a number of reasons, some of them, different measurement methods were used in collecting the data and different data centers were investigated. In spite of the difference, the point taken from these studies is that flows in data centers are short-lived and the depend on the applications dominating the data center of concern.

The arrival rates of flows found in these studies is also different: In [28], flows are recorded to be arriving at approximately 15ms of each other whereas in [27] flows which can arrive at a switch within 10μs of each other were reported. Differing reasons can be expressed for the large difference on the arrival rates, but, those reasons are unimportant for the purposes of the research presented in this report. The point to be taken into account is that flow arrival rates in order of microseconds or milliseconds are experienced in data centers. Thus, the flow durations and arrival rates of flows in data center networks are seen as the key attributes which makes data center traffic meet the traffic diversity requirement.

Another important attribute of data center traffic reported in the studies ([28], [27]) is the traffic distribution in the servers and racks. Both studies agree that approximately 80 percent of traffic is kept within a rack in data centers that run big data processing tasks such as map reduce. This attribute satisfies the requirement for topology aware traffic generation.

A number of network traffic generators have been published in the networking literature. The following traffic generators were investigated for the study under question: TCPReplay [78], D-ITG [79] and DCT^{2}Gen [80]. TCPreplay replays TCP traffic traces captured from a live network. The lack of publicly available traffic traces for data centers made it impossible to use TCPreplay for the research work presented in this report. Furthermore, it is also a difficult task to setup a network with similar parameters as the one from which the traffic traces were collected.

D-ITG generates traffic on a packet basis and the traffic can be set to follow known statistical distributions like uniform, exponential, gamma distributions and others. This technique is considered not realistic because it does not take into account real life conditions such as topology of the network as well as the applications (web search, map reduce etc.) generating the traffic. Further, the requirement of flow-based traffic generation stated above is not met by the D-ITG. Hence, all traffic generators which employ similar approaches as the D-ITG were considered not applicable for this research.

DCT^{2}Gen takes a different approach to traffic generation. It was designed specifically for the generation of traffic for data centers. DCT^{2}Gen makes use of the findings of data center traffic studies [28] and [27] to come up with a model that generates traffic on a flow basis. As result, traffic generated by the DCT^{2}Gen takes the attributes of data center traffic presented in the studies into account when genarating traffic. The important attributes taken into account include topology-awareness (number of hosts and racks); most flows are short-lived and most traffic is kept within the same rack. The generated traffic was evaluated and it was found that it meets the traffic properties reported in [28], [26], and [27], and for this reason the DCT^{2}Gen was considered acceptable for the realism requrement of the research. Moreover, DCT^{2}Gen is open-source, which makes the research results to be reproducible.

In order to generate traffic using DCT^{2}Gen, layer 2 network features are used as inputs and the generator outputs a layer 4 (TCP) traffic schedule (flows between origin-destination pairs) [80]. The layer 2 parameters required are the number of racks and the number of hosts per rack [81]. The TCP schedule generated is a .csv file containing flows defined by the following fields: source, destination, start time and the payload bytes [81]. The task of the researcher then becomes to find a method to replay the TCP traffic on a network prototype of their choice, in this research the task is to play the traffic on Mininet.

3.4 Playing Traffic in Mininet

Mininet Hi-Fi was designed to provide a robust network emulation platform on a laptop or any easily available computing device. However, a network whose computing requirements exceeds those available on a given laptop may degrade the performance of the emulated networking devices. For example, consider a situation where a Mininet host sends a packet at a time when all resources available on the processing device (laptop/computer running Mininet) are in use, the packet will not be forwarded until resources are made available [59]. In this situation, computational tasks which were meant to be processed in parallel end up being processed serially, which reduces realism in the emulator [59]. Therefore, replaying traffic in Mininet on a laptop requires one to perform experiments with this limitation in mind.

The topologies of choice in data centers are Clos-based and the most common one is a fat tree [73] [8] [74]. In efforts to reduce the computational requirements of the data center network, a fat tree of $K = 4$ is used. Figure 3.4 shows the fat tree topology used. The network consists of 20 switches, 16 hosts and 48 links. This is indeed a very small data center, but the size of the data center is not important in this research. The attribute of utmost importance is the realism of the traffic and the practicality of the network. It is critical that the forwarding of packets (and hence the generation of flows) in the computing device runnning Mininet is not hindered due to congestion in the resources of the computer hosting the network emulator.

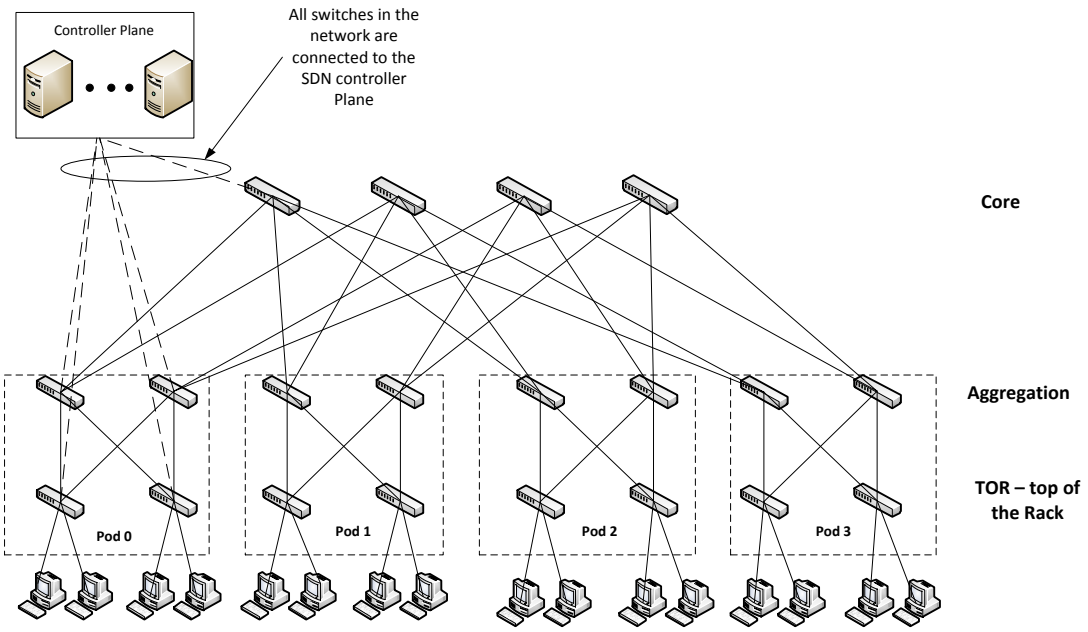


Figure 3.4: A Fat Tree Topology of $K = 4$.

3.4.1 Execution of programs in Mininet Hosts

The task at hand is to play traffic inside the Mininet hosts. This essentially entails to run a program inside a bash shell process which is attached to a network namespace [82] [83]. The Mininet software provides a user with the capability to run programs like "`ping`" and "`iperf`" inside Mininet hosts. The Python programming community and the Mininet API also provides several methods which can be used to interact with processes running inside shell processes. Popular methods in the Mininet API include "`cmd()/sendCmd()`", "`Popen()` and "`ssh`."

The `cmd()/sendCmd()` looked easier and promising in the early days of the research. A problem with this method was encountered when the number of communications between hosts was increased in efforts to emulate a data center network. It was discovered that when a process is running between two hosts through a `cmd()/sendCmd()` method, it becomes impossible to start another process between other hosts in the network through the same `cmd()/sendCmd()` mechanism. This mechanism is therefore blocking. In such a technique, the execution of programs inside Mininet hosts cannot take place in parallel. Essentially, IP packet flows could not be run concurantly in the network, which reduces the realism aspect of the network emulator. Hence, the

`cmd()/sendCmd()` method could not be applied in this research.

The Python pipe-based interface module which is implemented by the `Popen()` method was also considered. Unlike the `cmd()/sendCmd()` approach, the `Popen()` pipe-based approach supports multiple processes. Although this approach gives the capability to run processes in parallel, the task which remains cumbersome is to manage and monitor the outputs or results of these processes. The `Popen()` module provides a function called `pmonitor()` through which outputs of processes running in the background can be viewed. In this research it was discoverd that the use of the pipe-based approach suffers the same challenges as the `cmd()/sendCmd()` when the number of processes increases. The problem in the pipe-based approach is not the execution of multiple procesees, but the monitoring and management of the outputs generated by these processes.

The difficulty of using standard input/output/error and pipe-based approaches when running several processes concurrently has been reported in the programming literature. The Python Magazine issue of October 2008 included an article outlining the problems associated with the use of these approaches [84]. A method which in this research was found to outperform the aforementioned machanisms of accessing shell processes is the secure socket shell daemon (sshd) [85]. The attribute of sshd which finds great applicability in this research is the capability to fork a new daemon for every ssh connection [85]. This is a capability to establish multiple ssh connections concurrently. Consequently, by making use of the sshd capability available in the Mininet API and a multi-threaded software tool to run programs in the Mininet hosts, one can achieve a more realistic method of replaying traffic.

3.4.2 A Multi-threaded Software Tool to Replay Traffic

The traffic schedule generated by DCT^{2}Gen consists of the following fields: source host, destination host, flow start time and the size of the data to be transmitted [81]. This represents a TCP payload transmission between a source and a destination in the data center [81]. The traffic schedule generated by DCT^{2}Gen was scaled down to fit the size of the data center used in this research. In order to replay the traffic on Mininet, a multi-threaded software tool was developed by making use of the Mininet's Python

API. Each TCP flow in the schedule is executed by its own thread from a thread pool. This ensures that flows which run concurrently can be executed in parallel without having to be enqueued. The multi-threaded software was another effort to ensure that flows between the hosts are sent on time and in accordance with the times recorded in the traffic schedule. The DCT^{2}Gen paper [80] and the web page [81] does not specify a method to be used to replay the traffic schedule in a network emulator like Mininet. Therefore, the software tool used was developed specifically to meet the goals of this research.

The logic of generating and replaying traffic on Mininet by making use of the developed software tool is as follows:

- **Step 1:** Use DCT^{2}Gen to create a traffic schedule. The output of this file is a "`flows.csv`" file.
- **Step 2:** Load the `flows.csv` into the software tool which runs the flows between Mininet Hosts. The following steps describes the execution of the software tool.
- **Step 3**: Assign a thread to a flow and establish an SSH connection between the source and destination hosts.
- **Step 4:** Run a TCP `iperf` test between the source and destination with a payload size equal to that indicated in the traffic schedule.
- **Step 5:** Write the output of the `iperf` execution in a file. This step stores the results the text files generated by all the flows in a directory.

Step 3 to 5 are executed by different threads at the times indicated in the traffic schedule. These steps are repeated until the entire traffic schedule played. While data packets flow in the data path of the network (i.e. in Mininet), communications between Mininet and the controller are monitored by a network traffic analyser called Wireshark [86]. The traffic captured by Wireshark is then analysed by making use of statistical tools. The the logic and execution of the traffic replaying software is illustrated in Figure 3.5.

A distributed controller platform called ONOS [24] was selected for this research. This is because it is open source and guidelines for its usage are well documented. Any other

controller could have been used, as the objective of the research is not to evaluate SDN controller, but to study the workload seen by these controllers. In the proposal stages of the research it was planned that conditions which put a controller plane in congestion were going to be emulated or simulated. This is not seen as a requirement in this research, because increasing tha rate of flow arrivals at the switches with an intention for getting the controller to a state of congestion means altering the traffic schedule generated by the DCT^{2}Gen. With the traffic schedule altered, the traffic patterns seen by the controller plane might no longer match the data center traffic characteristics reported in [26], [27] and [28]. The goals of the research which are to determine the feasibility of forecasting controller plane workload and to determine the statistical features of the controller workload variable can be accomplished without getting the controller plane into a state of congestion.

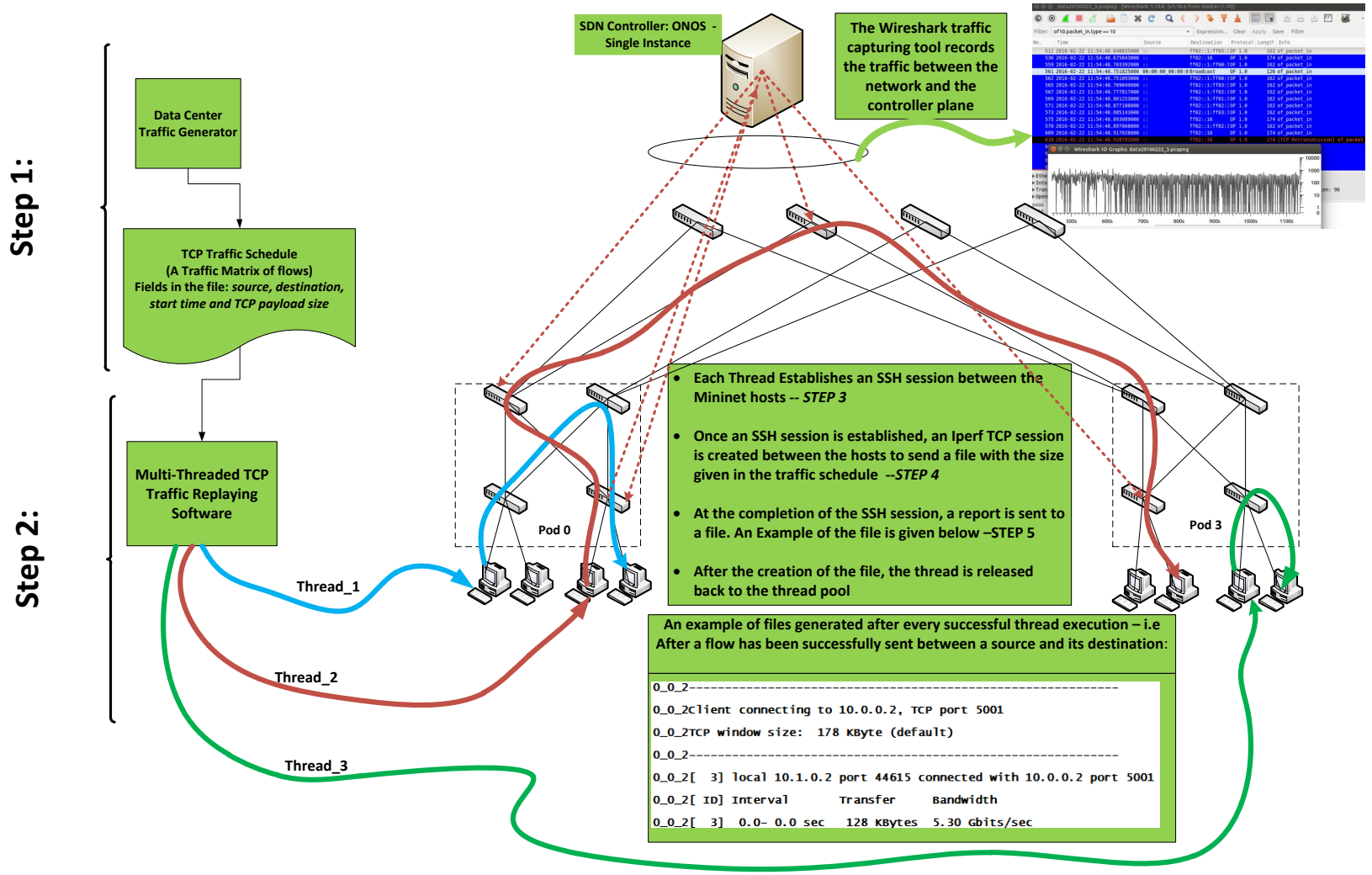


Figure 3.5: An Illustration of the Logic and Execution of the Traffic Replaying Software.

3.5 Chapter Summary

The research work under discussion requires the use of network emulation. Network emulators are preferred over testbeds and simulators because they can run on a low cost computing device like a laptop and produce results which are consistent with real hardware systems. The network emulator selected for the SDN-network experiment is Mininet. This is because it is open source and a widely used SDN-based network emulator. The ONOS controller plane was also selected for the same reasons as those applied in the selection of Mininet. A data center network was selected for this research because it has diverse traffic patterns and it is one network on which many researcher recommending and investigating the use of centralized controllers. A traffic generator suitable for data center traffic, DCT^{2}Gen, is selected for this research because it was found to be the only one that takes into account the topology of a data center in traffic generation. A new multi-threaded software tool was also developed to replay the generated traffic schedule on Mininet. It is believed that the network prototype developed for this experiment will meet the goals of the research.

4 CHARACTERISATION OF CONTROLLER WORKLOAD

The goal of this research is to explore the statistics of the traffic observed between the data plane and the controller plane of an SDN-based network. It is believed that the information which can be extracted from the traffic statistics can help in the design and operation of both controller plane software (such as Floodlight, ONOS, etc.) and SDN-based networks which use commodity SDN controllers. Two key outcomes are expected from this study: The first outcome is to identify the probability distributions and traffic patterns of the controller workload variable. The second outcome is to determine the feasibility of forecasting the controller plane workload. This chapter is about the characterisation of the controller plane workload variable, which is aimed at achieving the first outcome of this research. This is done by answering the following question:

- What are the probability distributions followed by the variable (or variables) which represents the controller-plane workload? Known probability distributions (Poison, Gamma, Exponential, Uniform, etc.) are fitted to the workload variable to determine the best candidate.

One of the most important steps in the design and development of a technology (or any product) is the testing step. This step involves exposing the newly designed product to the conditions it will encounter in its real life application. Network researchers face challenges when it comes to generating the network traffic patterns to which networking devices get exposed in real life deployment. For example, studies such as [26], [27] and [80] reported that traffic patterns seen in data centers are different from those seen in the Internet, WANs or LANs. Therefore, traffic patterns seen in the Internet are not applicable in the testing of networking devices designed for data center networks or vice-versa. These findings came after several studies were conducted on network traffic of data center networks and the Internet. Similar studies have not yet been undertaken

for the traffic seen by SDN controllers. It is for this reason that the first outcome (i.e. the first question above) expected from this research is to fill this gap.

The controller plane receives jobs through asynchronous messages such as `packet in` from the data plane elements (e.g. switches, routers, etc.). Therefore, The workload seen by the controller plane can be viewed in two aspects: the *arrival rate* of the jobs and the *inter-arrival times* between successive jobs. The arrival rate is applicable to answer a question like: "*How many jobs should be processed in a given period?*" The inter-arrival time may answer the question: "*How long will it take before the next job arrives?*" [28]. Given a multi-threaded controller plane, the first question could help determine the number of worker threads or controller instances required in a given period whereas the second question could be used to determine the order in which threads or controller instances may be scheduled. In teletraffic theory the arrival rate is referred to as *traffic intensity* and it represents the number of occupied resources [87]. The inter-arrival time is the time interval between successive calls (in telephone systems) or packets in a packet-switched network.

This chapter is arranged as follows: Statistical methods used in the characterisation of the controller workload variable are reviewed in Section 4.1. These are techniques used in the fitting of a given data set to theoretical probability distributions. Section 4.2 presents published research results against which the results obtained from this research can be compared. The set of data (i.e. the controller plane workload) obtained from this research is described in Section 4.3. The statistical properties of the observed controller workload variable are discussed in Section 4.4. The results obtained for the characterisation of the controller workload are evaluated in Section 4.5. A summary of the chapter is given in Section 4.6.

4.1 Fitting Statistical Distributions

Given a set of data or a random process which generates the data set, the selection of the probability distribution which best represent the random variable associated with the data set or the random experiment is called statistical distribution fitting [88]. The distribution fitting process also involves the estimation of the parameters of the selected probability distribution [88]. This report investigates the probability distri-

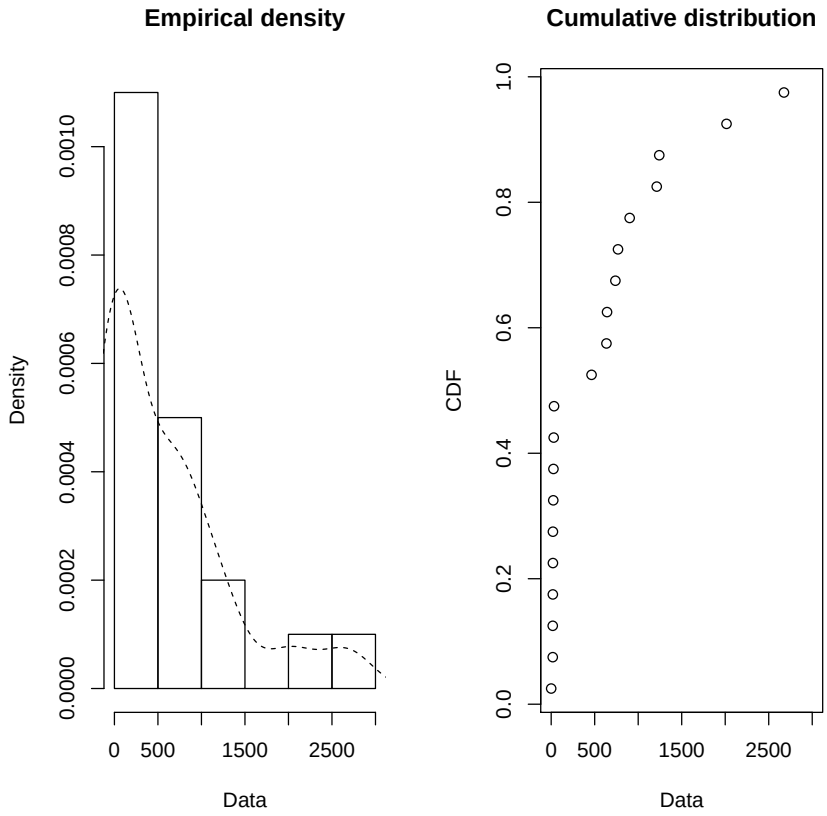


Figure 4.1: Example of Empirical Density and Cumulative Plots.

butions which are applicable to describe the controller plane workload in SDN-based networks. The following steps are followed in the process of distribution fitting: selection of candidate distributions, estimation of distribution parameters and assessment of goodness-of-fit.

4.1.1 Selection of Candidate Distributions

This requires an in-depth understanding of the process from which the data is generated [88]. Understanding the stochastic process is not a trivial task, and it is inadequate for one to rely on generally used assumptions about a data generating process. Therefore, the use of statistical techniques to guide this process is preferred in this research. This is because the statistics of the traffic seen by SDN controllers have not been extensively studied in the past and the results obtained from this research are expected to reveal these traffic statistics.

This research uses graphical approaches and descriptive statistics in the selection of candidate distributions. The graphical approach involves a plot of an empirical distri-

bution (e.g. an empirical cdf) followed by the given data set or random variable. The empirical distribution is then compared against theoretical ones. The candidate distributions are then selected from the theoretical ones which closely match the empirical distribution. Plots of the empirical distribution and the histogram of the data are generally the starting point of the graphical technique [88]. Figure 4.1 shows an example of a plot of a histogram, empirical density and cumulative distributions. These plots are used as a starting point in the search for candidate distributions.

Descriptive statistics based on skewness and kurtosis are also used to aid the selection of candidate distributions. The skewness of a distribution describes the symmetry of the probability distribution and the kurtosis describes the tails of the distribution. A symmetrical distribution is indicated by a skewness value of zero while a kurtosis value greater than three means that the given distribution has a heavier tail than that of the normal distribution [88]. The skewness and kurtosis for observations that are independent and identically distributed (i.i.d) are given by Equation 4.1 and Equation 4.2 respectively [88]. The equations of $\widehat{S}$ and $\widehat{K}$ are the corresponding unbiased estimators for the skewness and kurtosis respectively.

$$S(X) = \frac{E[(X - E(X))^3]}{VAR(X)^{\frac{3}{2}}}, \quad \widehat{S} = \frac{\sqrt{n(n-1)}}{n-2} \times \frac{m_3}{m_2^{\frac{3}{2}}} \tag{4.1}$$

$$K(X) = \frac{E[(X - E(X))^4]}{VAR(X)^2}, \quad \widehat{K} = \frac{n-1}{(n-2)(n-3)}((n+1) \times \frac{m_4}{m_2^2} - 3(n-1)) + 3 \tag{4.2}$$

Where X is the random variable which generates the observations; m_2,m_3,m_4 represents the empirical moments: $m_k = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x_i})^k$, where n is the number of observations, x_i represents the observations and $\bar{x}$ is the mean value of the observations.

In efforts to aid the use of descriptive statistics, the Cullen and Frey Plot [89] is used. This plot shows a graph of squared skewness and kurtosis with different points in the graph denoting the skewness and kurtosis values of theoretical probability distributions. Examples of distributions indicated in the Cullen and Frey plot include the normal, uniform, exponential, logistic, beta, log-normal, gamma and the Weibull distribution. An example of the Cullen and Frey plot is given in Figure 4.2.

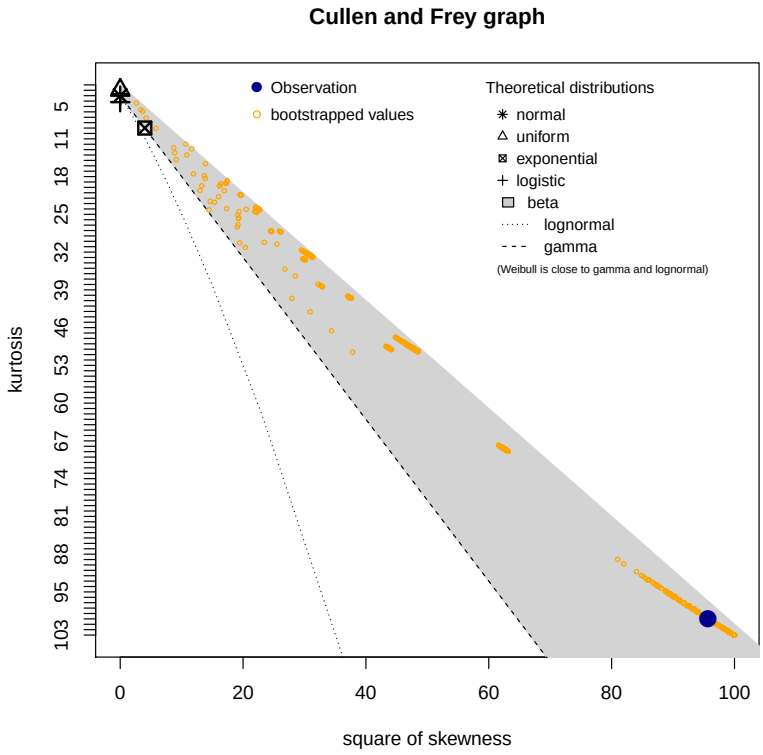


Figure 4.2: Example of a Cullen and Frey Graph.

4.1.2 Assessment of Goodness-of-fit

Once candidate distributions are selected, the fitting of distributions then begins. The methods used for this task are the maximum likelihood estimation (MLE), maximum goodness-of-fit estimation (MGE) and moment matching estimation (MME) [88]. These methods are used for the estimation of the parameters of the distributions. In this research, the software package *fitdistrplus* [88] in GNU R is used for the estimation of the model parameters. This package implements all these parameter estimation methods. Even though this is an automated process, an in-depth understanding of the statistical properties which are critical in the data set under observation need to be kept in mind. This means that the use of the R package still requires the user to consider the properties of the observed data and the objective of the model before selecting an appropriate estimation method.

Once distributions are fitted to theoretical ones, the tests of goodness-of-fit are then performed. A basic and one of the classical methods of goodness-of-fit assessment is the plot of the empirical CDF in the same set of axis with the candidate theoretical distributions. This is used to indicate the parts of the distribution that fits the data

well. The P-P plot and the Q-Q plot are also used alongside the CDF to test the goodness-of-fit. The Q-Q plot is preferred to describe the goodness-of-fit in the tails of the distribution and the P-P plot is preferred to asses the goodness-of-fit in the center of the distribution [88]. Furthermore, there are other standard statistical methods which are used in the assessment of goodness-of-fit. The ones widely used for continuous distributions are Cramer-von Mises, Anderson-Darling and Kolmogorov-Smirnov statistics [88]. The theory behind these methods is found in many statistics books and journal articles such as in [88]. These methods are also implemented in several R packages, including the *fitdistrplus* package.

4.2 Benchmarking and Comparisons of Results

A data center network was selected as a case study to be used in this research. Several studies have been conducted to reveal data center traffic characteristics. Benson *et al* in [26] and [27] and Kandula *et al* in [28] conducted extenisve measurements on data centers. These measurements were taken from data centers of different sizes (which also run different application) at universities, enterprises and other institutions. Other more recent studies on data center networks were conducted by Roy *et al* in [8] and Singh *et al* in [74], these were carried out in data centers used by social network and search engine companies.

The traffic generator used in this research [80] was developed from the data center traffic characteristics reported in [28] and [27]. For this reason, the traffic characteristics reported in [26] and [28] are used to test the credibility of the experiments conducted in this research. These data center traffic studies ([26], [27] and [28]) also report traffic statistics at a *flow* level. This makes it feasible for one to compare traffic charecteristics seen by a controller plane to the features of the *flows* reported in these data center network studies. Although the flows reported in these studies were not captured from a centralised controller, the attributes of the flows measured in switches are comparable to those of flows seen on centralized controllers. This is because the definition of a flow of packets is still the same (i.e. the basic attribute is the source and destination pairs). Therefore, the work of [26], [27] and [28] will be used in the evaluation of the results obtained in this research.

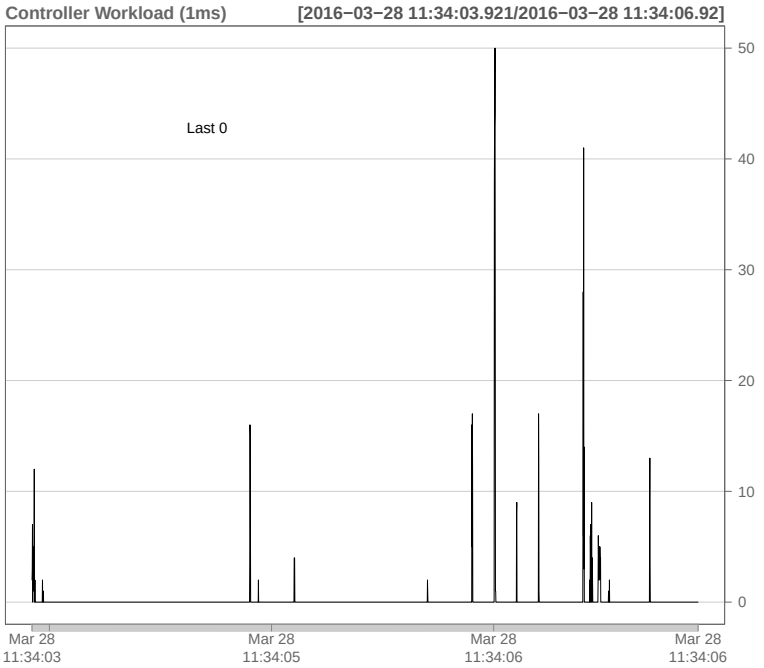


Figure 4.3: A Time Series of `packet in` Messages With Observations Made at 1ms Intervals.

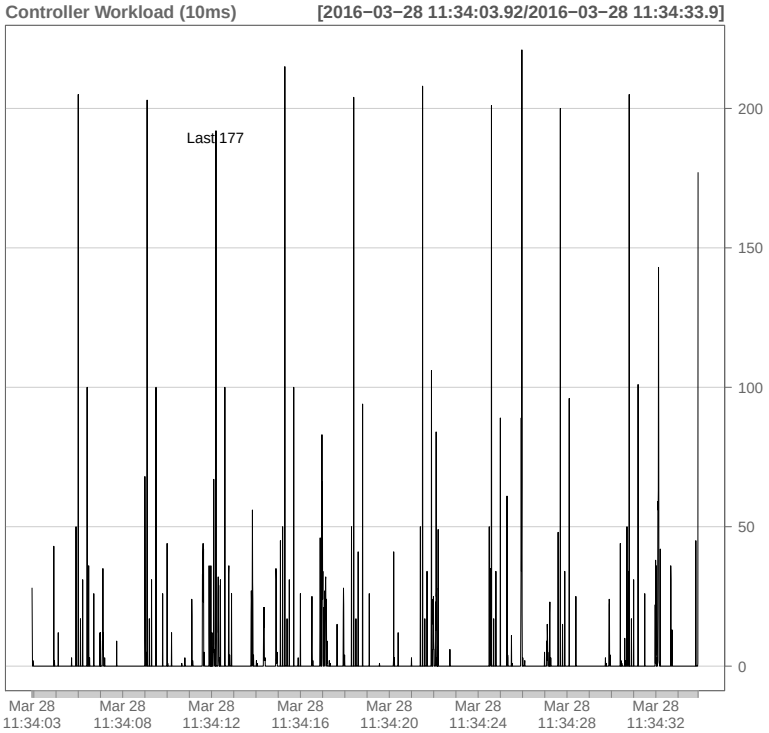


Figure 4.4: A Time Series of `packet in` Messages With Observations Made at 10ms Intervals.

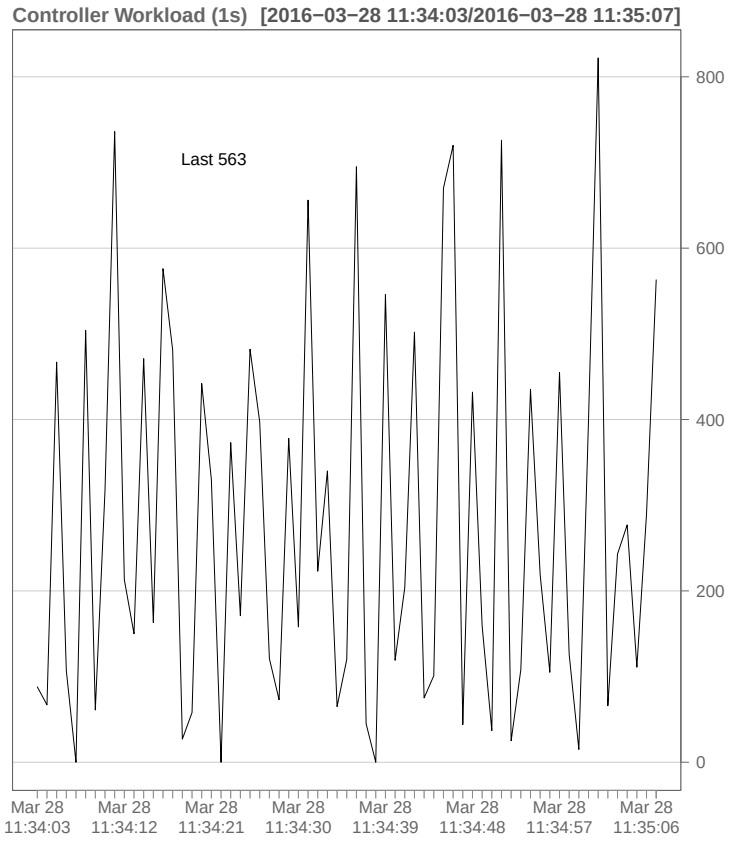


Figure 4.5: A Time Series of `packet in` Messages With Observations Made at 1s intervals.

4.3 Description of Observed Data

The variable of concern in this research is the workload seen by an SDN controller as data packets flow in the data path of the network. The types of conversations which take place between the controller plane and the data plane in an SDN-based network (OpenFlow 1.0) are discussed in Chapter 1, these are: *symmetric*, *controller-to-switch* and *asynchronous* messages. Asynchronous messages are the messages through which the data forwarding plane informs the controller plane of changes in state of switches; arrival of packets (flow initiations) or errors [3].

The workload seen by the controller due to traffic carried in the data plane is the arrival of new packets or new flows which need forwarding instructions. Apart from other tasks which the controller plane performs (such as network management), the task seen as the most critical in the scalability of SDN-based networks is the ability of the controller plane to successfully provide routing instructions to all legitimate packets

entering the network's data path. The workload is therefore seen as the arrival of new flows in the network. These flows are indicated by `packet in` messages which the data plane elements send to the controller upon the arrival of a new flow [3].

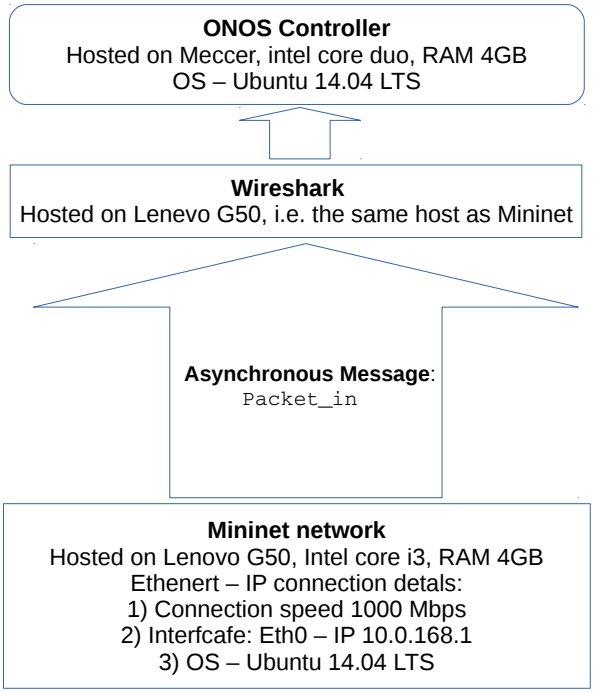


Figure 4.6: Data Capture Illustration.

The famous Wireshark [86] – a network protocol analyzer, was used to monitor and capture packets at the southbound interfaces of the SDN-based network. Wireshark was configured to capture packets at the network interface through which the Mininet network connects to the controller plane. Figure 4.6 illustrates the data capture experiment's setup.

The controller workload signal (`packet in` messages) was sampled at various granularities of time: 1ms, 10ms and 1s. This is the first step in the process to determine the traffic patterns seen by the controller plane. The characteristics of data center traffic as reported in [26] and [27] are expected to be revealed in this step. Hence, this section is also used to test the validity of the network prototype used in this research. Shown in Figures 4.3, 4.4 and 4.5 are the traffic patterns seen between the two planes with observations made at 1ms, 10ms and 1s intervals respectively.

In [26], it is reported that an ON/OFF traffic pattern was observed at the edge switches

of a data center network. Evidently the traffic patterns observed by the controller plane in this experiment is not ON/OFF. This points to the fundamental difference between the measurements taken in the two experiments: The experiment in [26] measured the amount of packets or flows received by a single switch while the experiment presented in this report measures the amount of flows received by a group of switches which are under the control of one controller plane. These measurements are taken from a centralised controller and *not* from the switches. Therefore, this proves that the workload seen by a device in its data-path is different from the workload the same device sees in its control path. In part, this finding also give the indication that traffic patterns previously used to test the performance of switches and routers may not be applicable in the testing of SDN controllers.

The 1ms time series (Figure 4.3) also shows that there are intervals of time where no asynchronous message is sent via `packet in` to the controller in request of routing instructions. This is expected because not all packet arrivals are sent to the controller (i.e. not all packets arrivals result in a flow initiation), but only those which do not find matching flow table rules in a switch's flow table. The graph also shows that the maximum time between the recorded flows is 20ms. This is in agreement with observations made by [27], where the majority of flows were found to last less than 100ms in a data center such as the one used in this study.

The time series of Figure 4.4 depicts the controller workload signal with observations made at 10ms intervals. This signal exhibits cyclic patterns where flow initiations or flow arrivals are seen as "groups" – i.e. a cycle of a group of flows is followed by another cycle and these cycles are aperiodic. This appears as though a group of flows gets initiated at the controller plane and after some time another group of flows is initiated. This is consistent with a data center network, for example, a web search application will send traffic to a number of hosts and those hosts may initiate communications with other hosts before returning a result to the original request. This type of traffic pattern is called the Scatter-Gather. In data centers which perform a number of map-reduce tasks or search related tasks, these type of patterns are prevalent [28].

Other critical factors which need to be taken into account in a network prototype are the imperfections of the system. Some studies have found that Mininet produces incorrect or inconsistent results under high traffic intensity [90]. Hence, the possibility

that flows may be missed due to failure to send packets by the Mininet software exists. Another source of packet transmission delays or failures by the network prototype may be due to the designed (i.e. designed specifically for this research) multi-threaded traffic replaying software. The system was designed such that threads from a thread-pool are scheduled to establish a flow between hosts in accordance with a traffic schedule which is generated by the traffic generator. In a situation where all threads are in use, the multi-threaded design becomes as good as a single-threaded software because a flow is essentially placed in a queue before it gets played.

These drawbacks in the network prototype are not seen as obstacles that will compromise the research result, but as realistic features of a data center network. This is because a data center networks also face interruptions due to network element failures. Studies like [91] indicate that even though data center elements such as top of the rack switches and aggregation switches are reliable, elements such as load balancers still cause failures to occur in the data center. The imperfections of the network emulation sotware are therefore seen as realistic features which a controller plane software is expected to come across in a data center network or any other real network. Hence, it is believed that the network prototype employed in this research is sufficient for use in the proposed research.

The observations presented in Figure 4.5 show a pattern of white noise. This is the controller workload signal recorded at 1s intervals. This signal appears like white noise because most flows in the data center last for less than 100ms, and as a consequence, sampling at 1s misses the true features of the observed patterns. The maximum number of flows recorded per second was found to be less than 1000. This may look unrealistically small when compared to what one could expect from a real life data center. This is because the simulated data center had 16 hosts in total as opposed to hunderds or thousands of hosts used in production data center networks.

4.4 Distributions of Inter-arrival Times

Statistical properties of network traffic are used in teletraffic engineering to model user demand [87]. The objective of this undertaking is to come up with tools and theory that can be applied in the design of efficient and cost effective communications

infrastructure [87]. Due to the lack of research work in statistical properties of traffic seen by SDN controllers, this section presents the study of fitting of the controller plane workload variable to known statistical distributions or mathematical models.

The traffic properties are divided into two stochastic processes: one which governs arrival of calls (or packets) and one which governs holding times (e.g. packet processing time) [87]. These random processes are assumed to be independent of each other [87]. This research investigates the controller plane workload, which is concerned about the arrival of `packet in` messages at the controller, and *not* the time the controller plane takes to process these packets. For this reason, the stochastic process of holding times is not significant in this study. However, the assumption of independence between the process of arrival of packets and that of holding times is upheld. This assumption is considered to be consistent with the principle of SDN, which states that `packet in` messages are initiated by the switches and are not influenced by the processing capability of the controller plane (i.e. these are *asynchronous* messages). The variables considered are therefore the traffic intensity and the inter-arrival times of `packet in` messages:

4.4.1 Flow Inter-arrival Times CDFs

Inter-arrival times are the time intervals between successive arrivals or occurances of an event of interest (e.g. arrival of calls or packets in a networking device) . In the controller plane workload problem, these are the time intervals between the arrivals of new flows or jobs to be processed by the controller. It is reported in [28] that the flow arrivals in data centers are non-Poisson, which means that the use of average inter-arrival times in designs of networks or networking devices is not valid. Therefore, this section investigates the distributions of the `packet in` messages received by the controller plane.

The maximum number of flows seen by the controller plane was found to be approximately 1000 flows per second (see Figure 4.5), over an observation interval of 1.067 minutes. In the observation window considered, the controller saw 24236 flows over a period of approximately 8 minutes. Data sets composed of 1000 successive flows were randomly selected from the observed 24236 flows. This was done iteratively to deter-

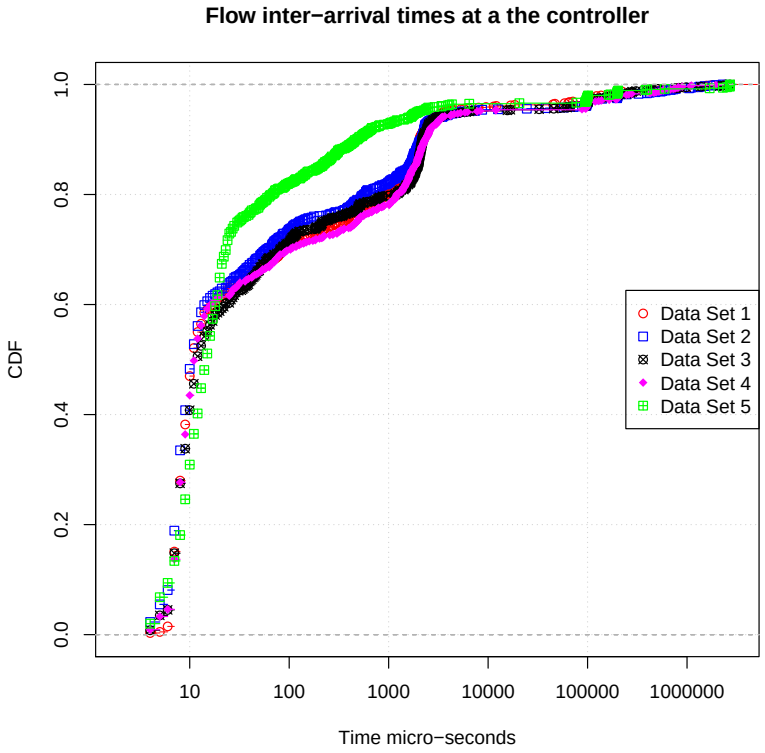


Figure 4.7: Flow Inter-arrival Times CDF.

mine the distributions of the flows. Shown in Figure 4.7 are the CDFs of five data sets - where each randomly selected data set consists of inter-arrival times of 1000 successive flows.

The CDFs of Figure 4.7 indicate that roughly 1-2% of flows in each of the data sets have inter-arrival times less than 10μs. This might appear to be in contrast to the flow inter-arrival times recorded in [27] and [28], where it was found that about 2-13% of the flows arrive within 10μs of each other. The reason for the larger percentage of flows which have inter-arrival times less than 10μs in [27] and [28] is twofold: The first reason is that the data centers considered in these studies ([27], [28]) are larger (500-10k hosts) when compared to the one emulated in this research which had 16 hosts. The second and more important reason is that the flow arrivals considered in [27], [26] and [28] were those seen at edge switches of a data center network. These flows are fundamentally different from those observed at an SDN controller, because a data plane element such as a switch handles all incoming flows whereas the centralised controller only handles the flows which do not have matching flow table entries at the switch. It is for this reason that switches experience a slightly larger number of flows which have inter-arrival times less than 10μs compared to centralised controllers.

The statistical information contained in Figure 4.7 also tell that about 80% of the flows have inter-arrival times between 10μs and $10m$s. This is in agreement with the inter-arrival times observed in [27], where 80% of the flows were found to have inter-arrival times less $1m$s. The noticeable difference between the two results is that the majority of flow inter-arrival times recorded in [27] occupied a narrower range of values (10μ – $1m$s) compared to those observed in this research (10μs –$10m$s). This is still linkable to the reason mentioned above, that is, the controller does not handle all incoming flows, but those which were not matched by existing flow table entries in the switch. Hence, a wider range of inter-arrival times (up to 10ms) was observed for the majority of the flows seen by the controller plane. However, it is important to note that the lower limit of the range of inter-arrival times (i.e. 10μs) of 80% of flows is the same on both the study conducted by [27] and the results obtained in this report. This is because the SDN-based network used in this research employs reactive forwarding, which entails that all new flows are sent to the controller. Therefore, the similar lower limit of 10μs accounts for a group of new flows which may be initiated at a start of a data center application; at the start of the network simulation or after a network element recovery (e.g. a link recovery). Alternatively, one may just think of these as new flow initiations whose first packets spend the minimal amount of time at the switches and end up at the controller plane for path computation.

4.4.2 Fitting Data to Known Probability Distributions

One of the tasks of paramount importance in the designs of communication networks and networking devices is to assess the capability of the network or device to successfully handle network traffic demands. In the case of this research, this task is to develop a traffic model that best fits the statistical characteristics exhibited by the controller workload variable. This entails the fitting of known statistical distributions to the recorded data. The commonly used mathematical software packages are MATLAB [92] and GNU R [93]. This research preferred the use of R, because it is an open source product that has several packages for both statistical distributions and forecasting. The packages used in GNU R are developed by different authors and use different mathematical methods. For example, the *fitdistrplus* [88] package for fitting distributions employs several methods to select the best candidate distribution for a given set of

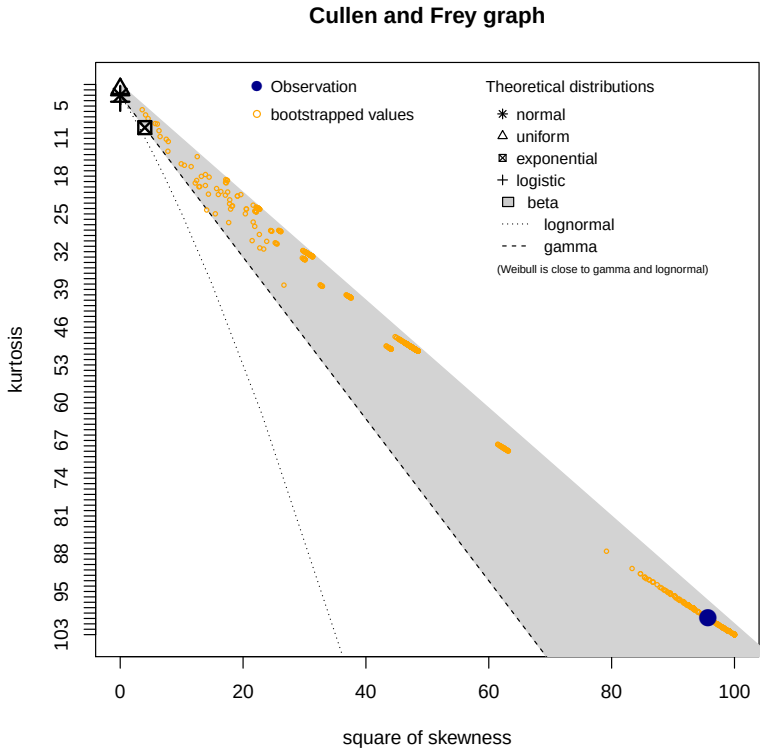


Figure 4.8: Skewness and Kurtosis Plot: One Hundred Flows.

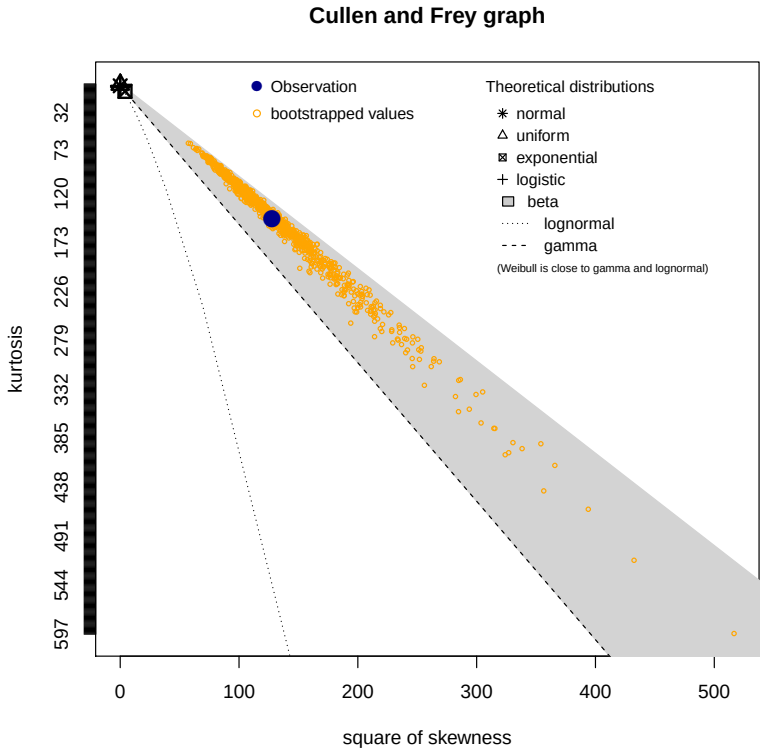


Figure 4.9: Skewness and Kurtosis Plot: One Thousand Flows.

observations including: maximum likelihood estimation (MLE), maximum goodness-of-fit (MGE) estimation and the moment matching estimation (MME). Hence, the user get to have a large pool of methods to choose from.

Shown in Figures 4.8 and 4.9 are the Cullen and Frey graphs which show the kurtosis and skewness of the empirical distributions of the controller plane workload variable. Evidently, the observed data set came from a family of beta distributions. In the graph of Figure 4.8, data sets of 100 flow inter-arrival times were used and in the graph of Figure 4.9, data sets of 1000 flow inter-arrival times were used. This is done to illustrate that skewness and kurtosis are unstable, which means that the candidate distributions selected by making use of the plots of Figures 4.8 and 4.9 need to be verified by making use of other statistical methods. It is important to note that both plots suggests a beta distribution which is close to a gamma distribution. This is evident by looking at the position of the observation point in the graph. Another critical feature of the data evident in the Cullen and Frey plots is the large kurtosis value. This indicates that the empirical distribution has a heavy tail, which may also indicate the presence of outliers in the data.

Network traffic distributions have been found to have heavier tails compared to the exponential distribution in several network traffic studies [94]. The tails of the distribution are of particular importance in this research: The left tails indicate the shortest inter-arrival times and the right tail indicate the longer ones. In other words, the left tails expresses high traffic intensity while the right tails describes less traffic intensity. In some studies, the tails may be associated with outliers or errors in the measurement system. This is not the case in this report, because both tails have a significant number of observations: 1-2 % in the left tail and about 3-5 % in the right tail.

Earlier in this chapter (Section 4.3) it was mentioned that the dominant traffic patterns in the data center used as a case study in this research are those attributed to the Scatter-Gether pattern, which is linked to map reduce applications. This means that a single flow may result in the initiation of several other flows, where the resultant flows may have inter-arrival times in the region of 80% of the flows (10μs – $10m$s inter-arrival times) or in the region of 1-2% (less than 10μs inter-arrival times) of the flows. Therefore, it is in part due to this reason that the tails of the distributions found in this report are not taken as outliers and should could not be removed from the data.

More importantly, the performance of the controller plane is judged by its ability to handle high traffic intensity. These may include flow initiations with inter-arrival times less than 10μs. This is basically the group of flows described by the left tail of the distribution. On the other hand, the flows described in the right tail of the distribution are as important as those in the left tail. For example, a network failure event may result in the recording of flows with longer inter-arrival times (e.g. 100ms), but the flows subsequent to the network failure event may have inter-arrival times less than 10μs. Therefore, it is clear that the controller is expected to handle network failures without disturbing the execution of network applications and also be capable of handling high flow initiation requests.

Essentially, both tails of the distribution are important in the evaluation of the controller's performance: the left tail is used to assess the controller's throughput and the right tails evaluates its ability to respond after events such as network failures or any other interruptions which may have caused a delay in flow initiation requests. The portion of the distribution where 80% of the flows lie is not considered important for the evaluation of the performance of the controller. This is because if the controller plane is capable of handling flows with inter-arrival times in order of microseconds, then the controller will be capable of handling flows with inter-arrival times in order of milliseconds.

In light of the aforementioned reasons regarding the importance of the tails of the distributions, a number of statistical distributions were investigated to fit the controller plane workload variable. These include the Weibull, gamma, Pareto, Burr, uniform, exponential and the normal distributions. The three methods (MLE, MME and MGE) of fitting distributions were also used in the search for a distribution that best fits the data. The MME was found to outperform the MGE and the MLE. The MME is sensitive to outliers [88], and for this reason, it is found applicable in the fitting of a distribution where the tails are important. The distributions that came close to describing the observed data are the beta and gamma distributions. This suggests that the Cullen and Frey plots were useful in this case.

Graphical goodness-of-fit measures are shown in Figure 4.10: the Q-Q plot and the empirical and theoretical CDFs. The Q-Q plot is preferred in this research because it stresses the nature of the fit with respect to the tails [94]. It is evident from the Q-Q

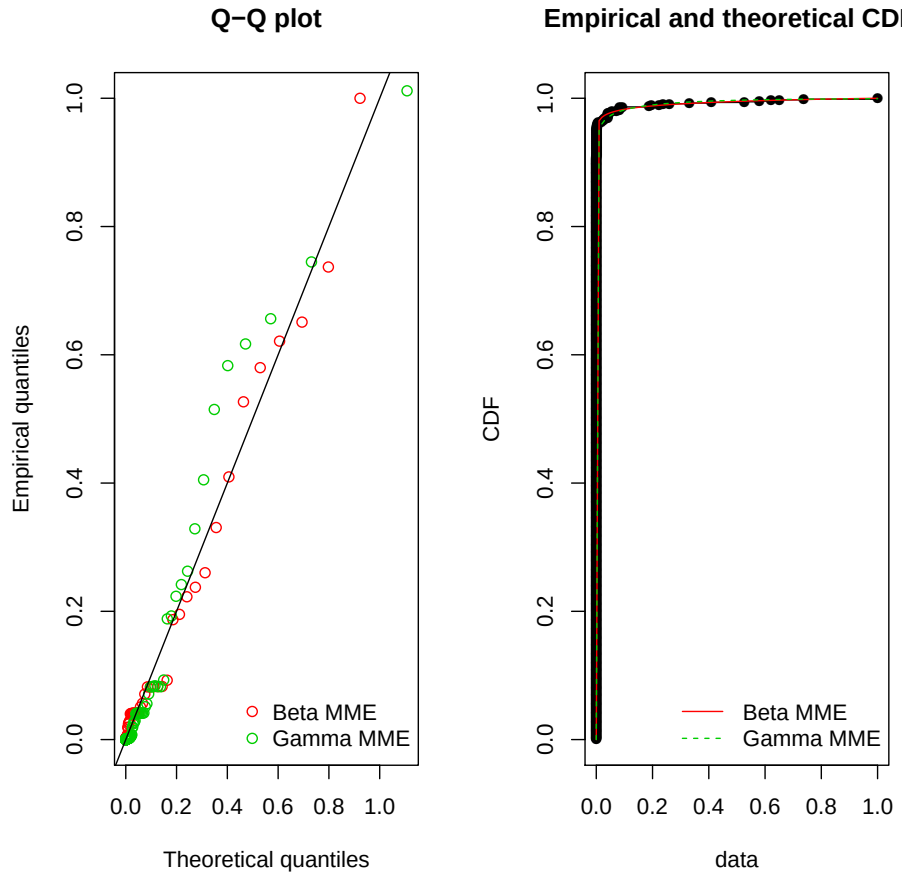


Figure 4.10: Fitted Gamma and Beta Distributions.

plot that the distribution which fits the tails best is the beta distribution. The CDF plot shows that both the gamma and beta are applicable in decribing the observed controller workload variable. It was also found that classical goodness of fit statistics: Kolmogorov-Smirnov and Cramer-von Mises favours the gamma dstribution. It is mentioned in [95] that making a statement about an entire distribution is different from making a statement about its tails. Therefore, for the purpose of capturing the behaviour of the tails, the beta distribution is preferred in this report.

Studies of arrival processes normally assume a Poisson distribution for modelling the random process of the arrivals. For examples, the Markov arrival process (M/M/1) for modelling a queue in networking devices assumes a Poisson arrival process where the inter-arrival times are modelled with a negative exponentially distributed random variable [96] [97]. In light of the extensive usage of Markov models and exponentially distributed inter-arrival times, one might assume that this research's results which suggests the use of a beta distribution for modelling inter-arrival times contradicts

the widely accepted use of Markov models. However, recent studies by the 3GPP in machine type communication (MTC) postulate that MTC devices which accesses network resources concurrently (or within a very short period) result in traffic flows that are highly synchronous and their inter-arrival times can be best modelled by a beta distribution [25].

Transmissions of data over a short period in MTC devices may be caused by a number of reasons, the one used by the 3GPP report is a scenario after a power outage [25]. After a power outage all MTC devices will send data to the network for re-establishing connections. Due to pre-programmed traffic forwarding rules in these devices, the resultant data transmissions are less random (the data transmissions occur in a synchronized manner) and occur within very short periods – i.e. bursty traffic flows [25] [97]. It appears that similar traffic patterns are seen by SDN controllers.

This research perceives two reasons for an SDN controller to see traffic patterns similar to those in MTC: First, OpenFlow compliant switches establish a communication session with an SDN controller in a similar way as MTC devices. This communication sessions may have to be established due to several reasons: at the start of the network, at the point where a new controller is added or after the recovery from a network error that broke a switch's connection to a controller. The second reason for the highly synchronous flows which are also bursty is associated with the data center traffic which is used in the network prototype. For example, a search engine's query sends out one flow which initiates flows to many other servers in the data center in search for results (i.e. the step which scatters in a Scatter-Gather pattern). In order to meet the search engine's time limits, the flows after the main request are initiated within very short time intervals. This results in bursty traffic flows seen by the controller. The same type of flows are also created in the step which collects the search engine's result from a number of machines (i.e. the gather step of the Scatter-Gether pattern). Therefore, the beta distribution identified to model the inter-arrival times of flows at an SDN controller is considered valid.

The network traffic literature on inter-arrival times which are modelled by making use of the beta distribution is still in its infancy [97]. Among the first few studies which studied these type of a traffic models are [25], [97] and [96]. The property of the beta distribution which is seen to be important in this research is the finite support

range, which is mentioned in [97]. This feature enables one to accurately capture the distribution of short inter-arrival times or bursty traffic flows. The negative exponential cannot model the short inter-arrival times as accurate as the beta distribution because of its (exponential) infinite support range [97]. Moreover, the successful modelling of short inter-arrival times by the beta distribution makes it applicable in evaluation of an SDN controller's throughput under high traffic intensity.

4.5 Evaluation of Results

The original question asked in the proposal stages of this research work was put across as follows:

(A) *Which statistical models forecasts the network traffic reasonably well to perform load balancing tasks such as resource allocation at the controller plane? Resources in the controller plane are the required controller instances in distributed controllers and computing facilities such as the number of threads or cores in a multi-threaded controller.*

It was realised as the research work unfolded that answering this question without an idea of the statistical properties of the controller workload variable would make it impossible or difficult for the researcher to understand the results of the forecasting techniques. This would be synonymous to a situation where one would like to forecast the amount of water a pipe supplies to a water tank over a specified time interval without taking into account factors such as the water source, the pipe's dimensions, the demand of water and others. Another example would be to describe a coin by considering only one of its sides. Therefore, in efforts to get a complete description of the traffic statistics of the controller workload variable, the following question was also explored:

(B) *What are the statistical distributions of the traffic between the controller plane and the data plane? The data is fitted to standard distributions such as log-normal, Poisson etc. to see if currently followed models are applicable in this setup.*

This question helps to fill another critical gap in the traffic engineering schemes currently used in the development of SDN controllers. Currently or at the time of the writ-

ing of this report, there were no studies which had explored the statistical properties of the traffic observed between the controller plane and the data plane of SDN-based networks with an intention to determine the feasibility of forecasting the controller plane workload. Even if one were to take out the intention to forecast traffic, studies which specifically studied the statistical properties of the flows between an SDN controller and its data plane devices were hard to find. The few studies which were found in this area of research are mentioned in Chapter 2 and the gaps which made it necessary for this question to be explored in this research are also discussed in Chapter 2.

The question answered in this chapter is question B. The statistical properties of the controller plane workload variable are discussed in Sections 4.4.1 and 4.4.2. The empirical CDFs exposed the distribution of the traffic as well as the typical behaviour of data center applications. It was discovered that 1-2% of flows arrive with inter-arrival times less than 10μs. More than 80% of the flows were found to have inter-arrival times in the range of 10μs–$10m$s. These flow arrival times were found to be consistent with measurements recorded in other studies of data center traffic properties. It was also discovered that the inter-arrival times at the tails of the distribution should not be taken as outliers because they represent realistic network behaviour such as link failures, bursty traffic patterns due to delay sensitive applications and others. The tails of the distributions were found to be usefull in the assessment of a controller's performance.

It was discovered that the traffic patterns observed between the data plane and the controller plane of an SDN-based network resembles those encountered in machine type communications (MTC). The beta distribution was found to be an appropriate model to describe the controller plane workload variable. This finding is considered novel in SDN-based networks, and consistent with the behaviour of network traffic generated by data center applications such as search engines. The finite support range of a beta distribution is seen as an advantage in modelling of inter-arrival times for data center applications like search engines. This is because such applications have stringent deadlines and the inter-arrival times of flows genarated by these application can be best modelled by distributions with a finite range like the beta distribution. The finding that the beta distribution models small inter-arrival times better than the widely used negative exponential is reported in [97], where the objective is to describe traffic patterns seen in MTC.

4.6 Chapter Summary

The objective of this chapter was to present the characteristics of the workload seen by SDN controllers. The importance of this statistical property in designs of SDN controllers was discussed. The first section of this chapter gave a review of the statistical methods which are applicable in probability distribution fitting. It was found that 1-2% of flows seen by the controller plane can arrive within 10μs of each other and more than 80% of the flows were found to have inter-arrival times in the range of 10μs - 10ms. The probability distribution which was found applicable to describe the inter-arrival times is the beta distribution. The finite range of a beta distribution was seen as a feature of utmost importance because it enables the accurate modelling of short inter-arrival times such as those seen in data center applications. Therefore, this means that the widely used Poisson arrival process (e.g. $M/M/1$) which assumes exponentially distributed inter-arrival times is not applicable for the modelling of inter-arrival times seen by SDN controllers. This is consistent with communication patterns seen in MTC. The statistical properties observed in the controller workload variable were also compared to those observed in data centers. It was concluded that the data observed by the controller in the experiment performed in this research is in agreement with data obtained in real life data centers.

5 FORECASTING OF CONTROLLER WORKLOAD

The utilization of network resources in an SDN-based network is a critical piece of information, which forms the back-bone of the network-wide view. The network wide-view is the attribute through which the logically centralised controller plane is achieved. More importantly, the network-wide view determines the scalability of an SDN-based network (as mentioned in Chapter 2). The current SDN literature on the composition of the network-wide view concentrates on the abstraction of data plane elements such as link utilization, port usage, switch flow tables and other resources. At the time of the writing of this research report, there were no studies that intended to represent the utilization of the SDN controller in a form of forecasts in the network-wide view data structure of controller planes. The second outcome of this report, therefore, aims at forecasting the controller's workload. This is done by answering the following question:

- Is it feasible to employ statistical forecasting methods to forecast the controller–plane workload? ARIMA and ANN are used to test the applicability of statistical forecasting in the workload seen by SDN controllers.

This chapter is about the forecasting of the controller workload variable, which in this research is defined as the arrival rate of `packet in` messages at the controller. The problem of forecasting the controller's workload or controller plane traffic intensity (i.e. the arrival rate) is approached as a time series analysis task. A brief introduction of time series analysis is given in Section 5.1. The theoretical background of Autoregressive Integrated Moving Average Models (ARIMA) and Artificial Neural Networks (ANN) forecasting models are discussed in Section 5.2 and Section 5.3 respectively. Section 5.4 discusses forecast evaluation techniques. Section 5.5 outlines the studies and benchmarks against which the research results obtained from this work is comparable. The results of forecasting using ARIMA and ANN is given in Section 5.6. Evaluation of the results is given in Section 5.7. A Chapter summary is given in Section 5.8.

5.1 Time Series Analysis

Time series analysis concerns itself with statistical and inference models which can be applied in the analysis of data that have been observed at different points in time [98]. This class of statistical modelling faces more mathematical constraints compared to traditional statistics. For example, the correlation introduced by observing data at adjacent points in time nullifies the applicability of the assumption that the observations recorded at adjacent points in time are independent and identically distributed (i.i.d) [98] [99].

A time series is defined as a sequence of random variables:

$$Y(t) = (y_1, y_2, y_3, y_4...). \tag{5.1}$$

This is a collection of random variables, $\{Y_t,\ t \in T\}$, where Y_t is the *time series* and t represents time. Hence, the variables, $y_1, y_2, y_3...$, in Equation 5.1 denote the values assumed by the random variable according to the order in which they are recorded in time. This research focuses on time series data in which the time intervals between successive observations of the random variables are equal. The property that the random variables are recorded at specific points in time means that these are *discrete-time* time series data, which are also known as *discrete-time* stochastic processes.

A complete probabilistic description of a time series (or a stochastic process) can be mathematically obtained by the evaluation of a joint distribution function [98] [99]:

$$F(c_1, c_2, ..., c_n) = P(y_1 \leq c_1, y_2 \leq c_2, ..., y_n \leq c_n). \tag{5.2}$$

The joint distribution F is evaluated as the probability that the time series observations are jointly less that n constants, c_1,c_2,...,c_n [98]. It is not necessary to explicitly deal with joint distributions such as the one given in Equation 5.2, because useful information can be obtained from the means, variances and covariances [99]. Therefore, the statistical properties of the data analysed in this research will be summarized in terms of the means, variances, covariances, and correlations (correlation and auto correlation

are of particular importance in forecast evaluation discussed in Section 5.4).

A time series model of observed data, $y_1, y_2, ...$, attempts to find a mathematical representation of the process from which the observed variables are generated. A variety of patterns can be observed from time series data, and the recognition of these pattern types play a vital role in the development of a time series model. Three main patterns are searched for in time series plots: *trend*, *seasonal* and *cyclic* patterns [100].

Trend: This is a pattern that exhibit long term increase or decrease over the period of observation [100]. As an example, time series data of power consumption of a city will show a long term increase as a city grows.

Seasonal: A time series pattern that is affected by seasonal factors such as end of the year, week, or even peak hours of the day [100]. An example of such a series will be Internet usage which is high during business hours and low at night.

Cyclic: A pattern that shows rises and falls of varying periods. The main difference between seasonal and cyclic patterns is that the rises and falls in a cyclic pattern are random while those in seasonal patterns are at specific times and have fixed durations [100].

In order to develop a statistical model about a time series, a number of assumptions may be made. The concept of *stationarity* of a time series is one of the most important assumptions which can be made about the values of a time series [99] [100]. A time series is said to be stationary if the statistical properties of the observed data do not depend on the time at which the observations are made [100]. This assumes that the process from which the observations are made is in statistical equilibrium [99]. Hence, a time series that exhibit seasonal or trend patterns is non-stationary while a series composed of white noise random variables is stationary [100] [98]. Mathematical methods used in the development of time series inference models take the stationarity requirement into account, as an example, ARIMA (discussed in Section 5.2) employs differencing to obtain a stationary time series.

5.2 Autoregressive Integrated Moving Average Models

The ARIMA approach is a multiple regression model where the dependent or explained variable is assumed to be a linear combination of past observations of the variable (i.e. regression of the variable against itself) [100]. Multi-variate forms of ARIMA do exist in literature, but the mostly used ARIMA model bases the forecasts on the history of the variable of concern only [101]. Therefore, for successful usage of this model, the forecaster should assume that there is continuity between the past and future values of a variable of concern [101]. One of the most important features of ARIMA which sets it apart from other time series forecasting models is the use of *autocorrelation* which is a measure of linear dependence between successive observations of a time series [101] [100]. This is also referred to as a *carryover* pattern in the data [101]. ARIMA has three components: *Autoregressive models*; *differencing* and *Moving average models*:

Autoregressive models (AR) uses the past values of a variable to predict its future values. It is this part of the ARIMA model which attempts to describe the carryover pattern between past and future values of a time series. The $AR(p)$ model is represented by Equation 5.3.

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + e_t. \tag{5.3}$$

Where $p = 1, 2, 3...$ is the number of past values to include in the model, c is a constant, y_{t-p} are lagged values of the time series, ϕ_p are parameters that affects the pattern of the series (different value of $\phi_1, ..., \phi_p$ gives different time series patterns) and e is white noise.

Moving average models (MA) uses past forecast errors to forecast the time series values. In the ARIMA model, the MA term is also referred to as the error feedback term and it uses past forecast errors to improve the forecasts [101]. An $MA(q)$ model is given by Equation 5.4.

$$y_t = c + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + ... + \theta_q e_{t-q}. \tag{5.4}$$

Where $q = 1, 2, 3...$ is the number of past forecast errors to include in the model, e_t is white noise and the θ_q values are parameters which affect the pattern of the time series (different values of $\theta_1, ..., \theta_q$ gives different time series patterns).

Differencing is one of many methods which are used to make a time series stationary. A differenced time series is defined as the difference between consecutive values of a time series. Equation 5.5 gives the mathematical definition of a differenced series:

$$y'_t = y_t - y_{t-1}. \tag{5.5}$$

Where y'_t is the differenced series. In case stationarity conditions are not met by applying differencing to the series once, the same procedure can be applied for a second time, which results in a second order differenced time series denoted by y''_t.

The $AR(p)$ and $MA(q)$ models can be combined through a differenced series and the resulting model is called the ARIMA model given by Equation 5.6.

$$y'_t = c + \phi_1 y'_{t-1} + ... + \phi_p y'_{t-p} + \theta_1 e_{t-1} + ... + \theta_q e_{t-q} + e_t. \tag{5.6}$$

Where y'_t is a differenced time series and the variables on the right-hand side of the equation represent both $AR(p)$ and $MA(q)$ components. This equation is called $ARIMA(p, d, q)$ where

- p = order of autoregressive part, which indicates the number of past values of the time series to be include in the model.
- q = order of moving average part, which indicates the number of forecast errors to be included in the model.
- d = degree of differencing, which indicates the number of differencing operations to be performed in the time series data.

The task of determining the values of (p, d, q) is normally carried out by making use of computer software packages. These software packages have been found to be more

effective compared to manual approaches proposed in literature. This research employs a GNU R package called `auto.arima` developed by authors of [100].

5.3 Artificial Neural Network

Artificial Neural Networks (ANN) are a class of statistical learning models which are used across a wide range of disciplines including engineering, computer science, finance, health sciences and others. These fields employ ANN to solve problems such as function approximation; classification; pattern recognition and time series prediction. ANN models do not make mathematical assumptions about the data generating process. This is one of the reasons ANN models are considered superior to other models which are used for non-linear function approximation [9]. The learning process of ANN relies solely on the provided data and also adapts to updates in the input data. For example, the ARIMA model described above employs differencing to ensure the stationarity of the time series in question whereas ANN relies on the given data to approximate the process from which the time series data is generated.

The basic idea behind ANN is to compute a linear combination of the input features (i.e. the original data) and then approximate the target function as a non-linear function of the linear combination of the input features [102]. An ANN structure consists of multiple layers of neurons which are arranged such that the input features are input at the first layer and the output is produced at an output layer. In a basic single neuron, the computed linear combination of inputs is used as an argument to a non-linear activation function.

The non-linear activation function is also called a squashing function, because it takes a real-valued input and produces a value between 0 and 1, and it also has to be a non-decreasing and differentiable function [103] [102]. Commonly used activation functions include the sigmoid function and hyperbolic *tan* whose mathematical definitions are given by Equations 5.7 and 5.8 respectively.

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{5.7}$$

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}. \tag{5.8}$$

The widely used ANN model for time series forecasting is the single hidden layer feed-forward network shown in Figure 5.1 [9] [100]. This is because with a single layer, a linear model can be developed and with the addition of a hidden layer both linear and non-linear models can be created. Equation 5.9 presents the relationship between the inputs and outputs of the network shown in Figure 5.1.

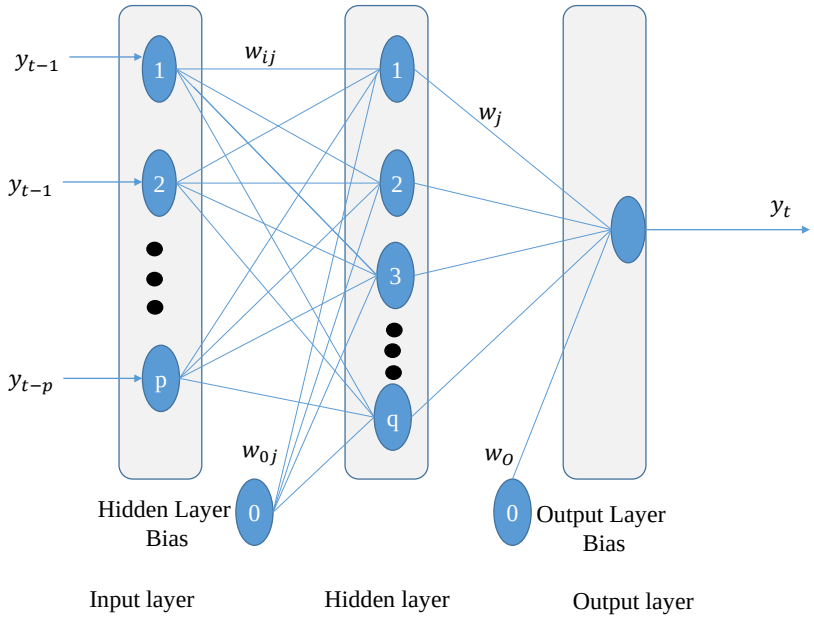


Figure 5.1: An Illustration of an ANN Model – Adapted From [9].

$$y_t = w_0 + \sum_{j=1}^{q} w_j . g(w_{0j} + \sum_{i=1}^{p} w_{ij} . y_{t-i}) + \varepsilon_t. \tag{5.9}$$

Where y_t represents the forecasts (predicted variable); $y_{t-i}...y_{t-p}$ are past values of the predicted variable; w_{ij} are weights between the input and hidden layer; w_j are weights between the hidden layer and the output layer (the weights are the ANN model's parameters); $i = 0, 1, 2..., p$; $j = 0, 1, 2, 3..., q$; p is the number of input layer neurons; q is the number of hidden layer neuron; the function $g(.)$ is an activation function and ε_t is a stochastic component which is generally assumed to be white noise [9]. An ANN network is normally trained to determine the weights (w_j and w_{ij}). A standard

method used in the training of ANN is the *backpropagation* algorithm and *gradient descent* procedures [104]. These techniques are explained in a number of machine learning books such as [104].

The activation function of each of the neurons in the network depends on the layer on which the neuron is found. The neurons in the input layer do not apply any activation function as their role is just to transfer inputs to the hidden layer [9]. The hidden layer neurons do apply the activation function while the output layer does not apply any activation function. Consequently, the single hidden layer ANN is a non-linear autoregressive model [9]. In the literature there is no straight-forward method for determining the values of p and q. It is, however, generally agreed that a simple structure with a small number of nodes performs fairly well for forecasting problems.

ARIMA and ANN models will be applied separately to the problem and the results from both techniques will then be compared. The objective of the exercise is to determine if it is feasible to forecast the controller workload and to determine which method forecasts well when compared to the other. The two techniques are selected because they are the basis of many other complicated forecasting approaches: ARIMA represents a class of linear models and ANN represents a class of non-linear models. ARIMA is also considered a benchmark to determine if one can obtain forecatsts for a for a given time series data. For this reason, there is no need to implement a combination of the two methods as it was done by the authors of [9]. This is because the question of determining the applicability of time series forecasting techniques in SDN controller plane can be answered through the basic ANN and ARIMA methods.

5.4 Forecast Model Evaluation and Forecast Accuracy

An optimal forecast is obtained through the use of a model that uses all the statistical information available in the provided (historical data) data to compute its forecasts. The forecast errors obtained from the model should not contain information which can be used to improve the forecasts [101]. It is, therefore important to point out that an optimal forecast does not mean the forecasts are accurate, but it means all the information provided by the input data is "fully" used by the model to compute its forecasts. Thus, in this research there are two evaluations that are followed: the

evaluation of the model and the evaluation of forecast accuracy.

The forecast accuracy is evaluated by making use of genuine forecasts: First, a model is trained to fit the historical data provided, and then, the model is tested with new data which was not used in the training set [100]. This method is widely used in machine learning, where the given data set is split into two parts, a training set and a test part. The norm is to train a model with about 80% of the data and test its accuracy on the remaining 20%. The data to be used in this research will be captured at different granularities of time, that is, at different orders of the time unit (10ms, 100ms, etc.). Therefore, both scale dependent and scale independent accuracy measures will be used.

Given a time series forecast where y_t is an observation made at time t and $\hat{y}_t$ denotes the forecast value for y_t, the forecast error is defined as $e_t = y_t - \hat{y}_t$, which is in the same scale as the data [100]. This means that forecast accuracy measures that are based on e_t are scale-dependent and can only be used to compare forecasts which are made on the same data [100]. Two scale-dependent accuracy measures which will be used are the mean absolute error (MAE) and the root mean squared error (RMSE) given in Equations 5.10 and 5.11 respectively.

$$MAE = mean(|e_t|). \tag{5.10}$$

$$RMSE = \sqrt{mean(e_t^2)}. \tag{5.11}$$

Where the *mean* function represents the sample mean, which is defined as $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$.

A solution to compare forecast accuracy across time series data points which are on different scales is proposed in [100]. This method uses a scaling statistic "Q" which is computed on the training data [100]:

$$Q = \frac{1}{N-1}\sum_{j=2}^{N} |y_j - y_{j-1}|. \tag{5.12}$$

Where N is the length of the training data set, y_j and y_{j-1} are consecutive values of

the time series. This equation basically computes the MAE for a naive forecast. A naive forecast is a forecasting method where the one step ahead forecast is taken as the previously observed value of the time series [100]. The mean absolute squared scaled error is given in Equation 5.13.

$$MSAE = mean(|q_j|) = MAE/Q. \tag{5.13}$$

It is also important to point out that if a scaled error computed by making use of Equation 5.12 is less that one, then it is considered to have been obtained from a better forecasting model than an average naive forecast applied on the training data, and if it is greater that one it is considered to have to have been obtained from a forecast model worse that an average naive forecast applied on the training data [100]. The scaling statistic given in Equation 5.12 is valid for non-seasonal data, one which is applicable for seasonal data can be obtained in [100]. A similar forecast accuracy measure which is also widely used is the Theil's U statistics given in Equation 5.14:

$$U = \frac{\sqrt{\sum_{i=1}^{N}(y_i - \hat{y_i})^2}}{\sqrt{\sum_{i=1}^{N} y_i^2} + \sqrt{\sum_{i=1}^{N} \hat{y_t^2}}} \tag{5.14}$$

Where y_i and $\hat{y}_i$ are the actual value and the forecast respectively and N is the number of forecasts. A value of U lesser than one indicates that the model is capable of achieving high degrees of accuracy, whereas a value of U greater than one indicates that the model will produce forecasts which are worse than both guessing and the naive forecast method. U which is equal to one means a forecast model which is as good as the naive forecasting model. The MASE is preferred in this research work, because it is applicable for time series data that is on different scales, and more importantly it can be used to determine if a given model beats the naive forecast.

The evaluation of the optimality of the forecast model is carried out by making use of residual analysis. A residual is the difference between an observed value and its forecast [100] [101]. An optimal forecast model is defined as one which produces residuals which are uncorrelated and have a mean of zero. In case the residuals are correlated, then the residuals contain additional information which can be used to improve the

forecasts, and if the residuals have a non-zero mean, the the forecasts are considered to be biased. This means the residuals need to form a white noise series to ensure an optimally forecasting model. A test for correlation is performed by making use of the portmanteau test and the autocorrelation function plots (ACF). The portmanteau test is used to test for a group of autocorrelations while the ACF plots are used to check for autocorrelations between adjacent points of a time series.

The type of portmanteau test used in this research is the Ljung and Box test [105]. This test is based on the statistic Q^* given in Equation 5.15:

$$Q^* = T(T+2)\sum_{k=1}^{h}(T-k)^{-1}r_k^2. \tag{5.15}$$

Where T is the time series data's length, r_k is the autocorrelation of lag k, and h represents the number of lags used in the test [100]. A large value of Q^* indicates that the autocorrelations in the residuals are significant and therefore the estimated model needs to be improved. The value of Q^* is tested against a χ^2 distribution with $(h-K)$ degrees of freedom (df), where K is the number of parameters in the model estimated [100]. A method adopted to quantify the significance of the autocorrelations in the residuals is to observe the p-value obtained from the test: A value of p greater 0.05 is considered large enough to indicate that the autocorrelations are insignificant and can be ignored while a value of p less than 0.05 is considered small, which means that the effects of autocorrelations *cannot* be ignored [106].

The results of the ACF and the portmanteau test play a vital role in the calculation of the forecasts' prediction interval. The forecasts basically attempt to predict the value a random variable will take in the future. Therefore, the prediction interval is the range of values which the random variable could take with a specified probability [100]. The prediction interval is calculated based on the distribution of the forecast errors, if the errors come from a white noise signal, then the 95% prediction interval is given by Equation 5.16:

$$\hat{y}_t \pm 1.96\hat{\sigma} \tag{5.16}$$

where $\hat{y}_t$ is the forecast value of y_t and $\hat{\sigma}$ is an estimate of the standard deviation of the forecasts' distribution [100]. In case the forecast horizon is one step ahead, then the standard deviation of the residuals is approximately equal to that of the forecasts' distribution [100]. In this research the prediction interval will be calculated for forecast models which are capable of producing optimal forecasts.

5.5 Benchmarking and Comparisons of Results

The second outcome of this research is discussed in this chapter. This is to determine the feasibility of forecasting the controller workload variable. At the time of the writing of this research, there was no study that was found to have attempted to forecast the workload of a controller plane in SDN-based networks. However, many studeis have conducted the forecasting of network traffic: In Stolojescu-Crisan [107], ARIMA and ANN models are used to forecast network traffic in wireless networks. Cortez *et al* in [108] presents the forecasting of internet traffic using ANN. The Cortez paper also compares the forecasting power of ANN against ARIMA and Holt-Winters (a method of forecasting based on exponential smoothing). Another study which compares a wide range of forecasting models for internet traffic was conducted by Katris *et al* in [109]. This study investigates ARIMA, ANN, Holt-Winter and others. The network traffic forecasting studies mentioned above ([107], [108] and [109]) come to a similar conclusion, that is, taking non-linearity into account in forecast models leads to better forecasting [109]. This means that the ANN aproach is generally preferred when it comes to network traffic forecasting. This finding is taken into account in the research under discussion.

The science of statistical forecasting state that it is infeasible to compare the accuracy of forecasting models unless the same data set is used by the forecasting models of concern[110] [111]. The primary objective of this research is to determine if the traffic seen by controllers in an SD-based network can be forecast and *not* to come up with the best forecasting model. The naive forecast model is used as the most basic benchmark to determine if a forecasting model adds value to a process [112], [113] [111]. A model that performs worse than a naive forecast is considered a waste of resources while one that performs better than a naive forecast is used to tell if forecasting techniques will add value to a given process [112]. It is mentioned in [113] that it can be hard

to find a forecast model that beat the naive forecast model (this also depends on the given data). Therefore, before improvements can be made on a given statistical forecasting model, the model should perform better than the naive forecast. Therefore, this research uses the MASE accuracy measure proposed by Hyndman *et al* in [114] to determine the feasibility of forecasting the controller workload variable. This forecast accuracy measure can be used to determine if a model performs better than the naive forecast. Therefore, comparisons between ARIMA and ANN models are made for the cases where the developed models perform better than a naive forecast.

The forecast horizon is another critical attribute in the task of determining the feasibility of forecasting the controller workload. In this research the forecast horizon should be long enough to enable an application running at the controller plane to allocate controller plane resources. For example, if the forecast horizon is 1s ahead, then the 1s should be long enough for the network application to add controller plane instances. Therefore, the question of "forcastability" of the controller plane workload is answered by determining if a developed forecast model beats the naive forecast and if the forecast horizon is long enough for the controller plane to make controller plane resources available.

5.6 Forecasting Traffic Intensity

Traffic intensity or the controller-plane workload is the rate of arrival of new flows or the arrival of path computation requests at an SDN controller. The forecasts of the traffic intensity signal are aimed at predicting the amount of flows which the controller plane needs to anticipate in a given period of time. The ultimate goal of forecasting the controller's workload is to establish a representation or abstraction of the state of the controller plane in the coherent network-wide view which is kept in the SDN controller plane. This representation of the state of the controller plane in a form of forecasts will enable applications running on top of the controller plane to schedule resources in accordance with network traffic demands.

The statistical forecasting exercise is done in two steps: The first step involves the development of a statistical model that will be used to generate forecasts. This model is then evaluated by making use of the residual examination tools mentioned in the

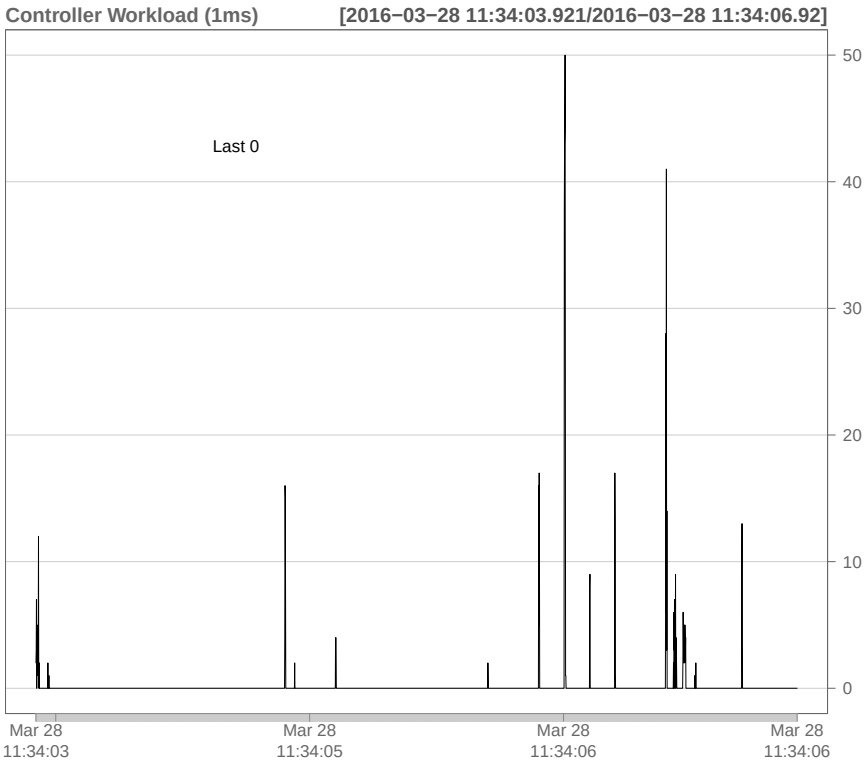


Figure 5.2: A Time Series of `packet in` Messages With Observations Made at 1ms Intervals.

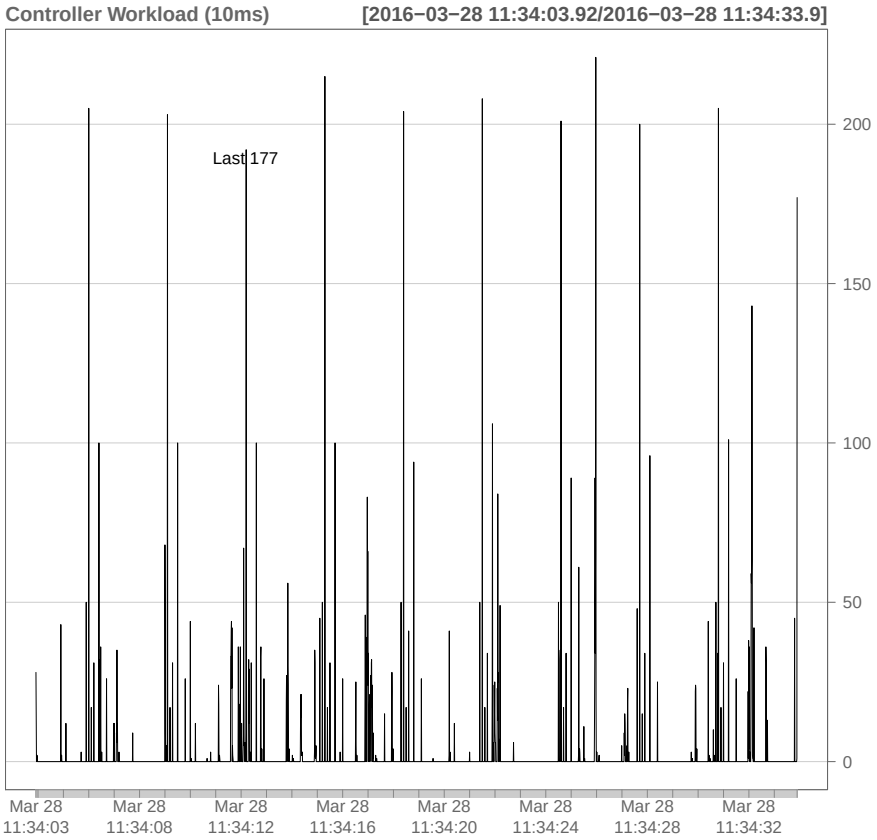


Figure 5.3: A Time Series of `packet in` Messages With Observations Made at 10ms Intervals.

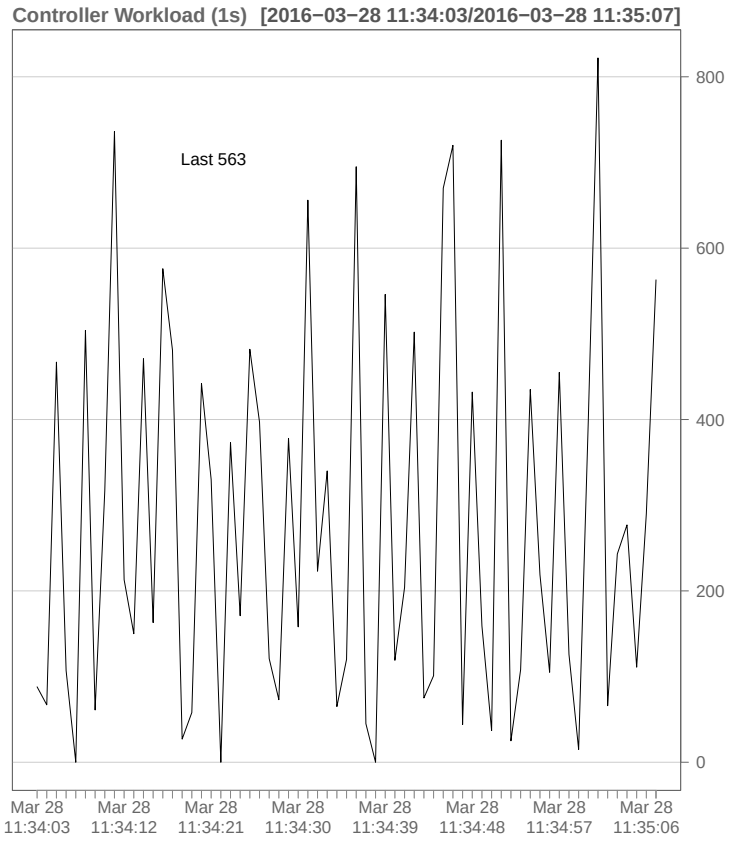


Figure 5.4: A Time Series of `packet in` Messages With Observations Made at 1s Intervals.

preceding section. The second step is concerned about the accuracy of the forecasts. Forecast accuracy measures used are: ME, RMSE, MAE and MASE. Graphical plots of the forecasts and the true time series data are also used to compare the forecasts and the correct time series data. The aim of the research work presented in this report is not to develop a new forecasting method or to improve existing ones, but it is to determine the feasibility of forecasting the SDN controller plane workloads.

The controller plane workload variable or traffic intensity seen by the controller plane is depicted in Figures 5.2, 5.3 and 5.4. The time series plots presented in Figures 5.2 and 5.3 are a specific class of time series variables, called a *time series of counts*. A time series data where the value of the series at time index t is denoted by y_t and y_t takes on small non-negative integer values is called a *count time series data* [115]. This type of time series data arises in situations where the occurrence of an event is counted over a given period of time [116] [117].

Most count time series data contains many periods of zero or the time series takes on the value of zero more than other all the values which it can take. Hence, the time

series of Figures 5.2 and 5.3 are specifically referred to as *zero-inflated time series data*. The forecasting body of research knowledge also refers to count time series as *discretely* valued time series data [115]. This is because such a time series variable takes on a set of integer values ($y_t \subseteq \mathbb{Z}$) instead of a set of real numbers ($y_t \subseteq \mathbb{R}$). Since the property that a time series is zero-inflated can be easily seen in time series plots (e.g. Figures 5.2 and 5.3), this research report uses the term "zero-inflated" instead of "discretely valued" when referring to the low count time series of Figures 5.2 and 5.3. Many researchers in the field of statistics have mentioned that forecasting methods for continuously valued time series such as ARIMA are not applicable to zero-inflated time series data or low count time series problems [116] [118] [115]. Therefore, the forecasts of the time series of Figures 5.2 and 5.3 are computed with this claim in mind.

Unlike the zero-inflated time series data depicted in Figures 5.2 and 5.3, the time series presented by Figure 5.4 is not zero-inflated. This research makes the assumption that the values which this time series (i.e. Figure 5.4) takes on come from a continuous random variable. This means that if y_t denotes the value taken by the time series at time, t, then $y_t \subseteq \mathbb{R}$. This assumption enables the application of methods such as ARIMA which assumes that the time series values are produced from a *white noise* random variable. Moreover, the assumption that time series values come from a continuous random variable is consistent with the basic definition of a time series given in a number of statistics books including [98], [100] and [99]. However, it is important to note that discrete random variables are also applicable to characterise time series data [115]. For examples, the count time series data described in [115] are modelled by discrete random variables.

This research makes two assumptions about the data observed by the SDN controllers: Firstly, time series data which is zero-inflated is assumed to be a count time series and it takes on integer values. Secondly, a time series which is not zero-inflated is assumed to be able to take any real number at the time when an observation is made, these type of time series are called *continuously-valued time series data*. It is also important to note that all the time series data considered in this report are *discrete in time*, because the observations are made at specific time intervals (as mentioned in Section 5.1).

5.6.1 Auto-Regressive Integrated Moving Average Models

The controller workload plots shown in Figures 5.2, 5.3 and 5.4 represent the traffic intensity. There is no seasonal or trend patterns observable from these plots. Thus, the time series variables depicted in these figures are stationary, which allows for the application of ARIMA(p,d,q) model for stationary time series variables.

5.6.1.1 ARIMA Forecasts of Traffic Intensity at 1ms Intervals

Flow arrivals captured at intervals of one milliseconds were recorded for 2.998 seconds [11:34:03.922 -to- 11:34:06.920], where 2299 observations were recorded. A forecasting model was developed as follows: A training set was selected to be the first 2250 records and a test set consisted of the last 749 records.

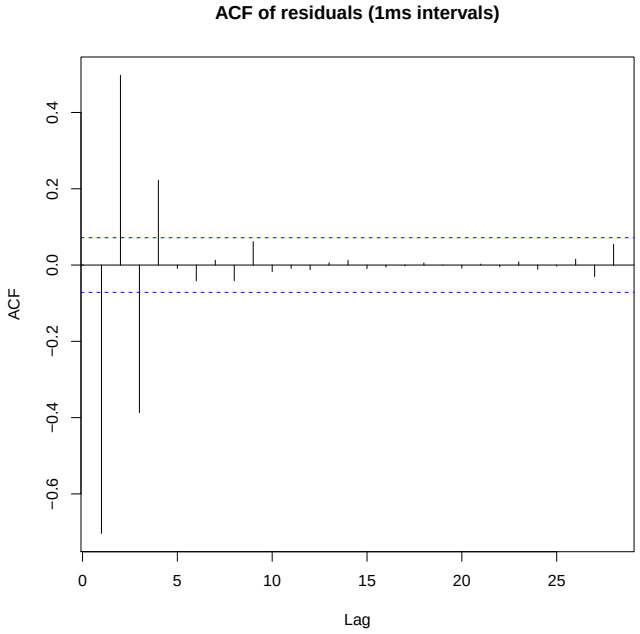


Figure 5.5: ACF Plot of Residuals – ARIMA Model Developed for the Time Series of 1ms Intervals.

Table 5.1: Results of a Portmanteau Test – ARIMA 1ms Time Series.

$h = 24$	$\chi^2 = 715.28$	$df = 28$	$p < 2.2e-16$

Evaluation of the trained model by making use of residual analysis showed that the residuals were not from a white noise signal. This is indicated by two residual analysis methodologies: First, the autocorrelation plot (ACF) is shown in Figure 5.5. Second, a *portmanteau* test whose results are shown in Table 5.1.

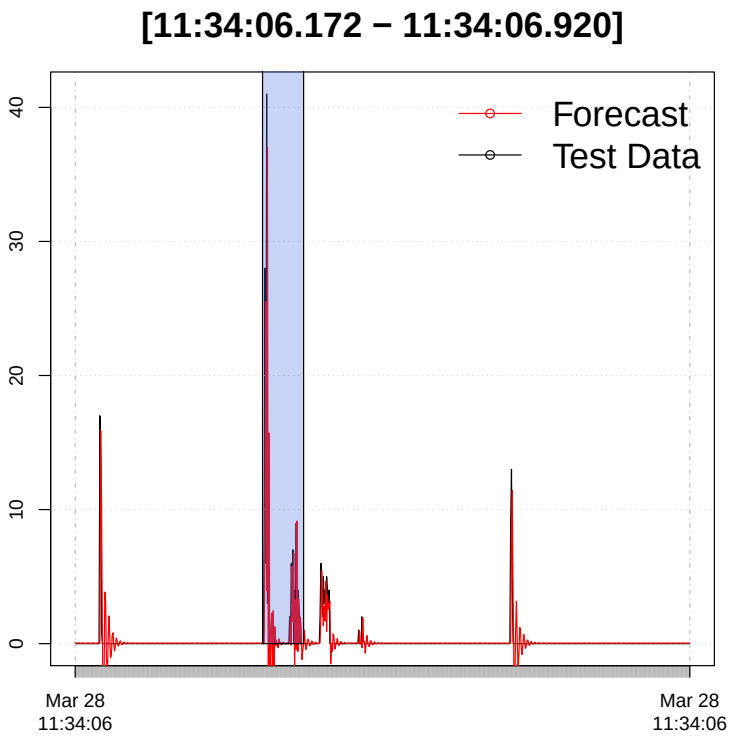


Figure 5.6: Time Series Plot on the Entire Test Data – ARIMA Model Developed for the Time Series of 1ms Intervals.

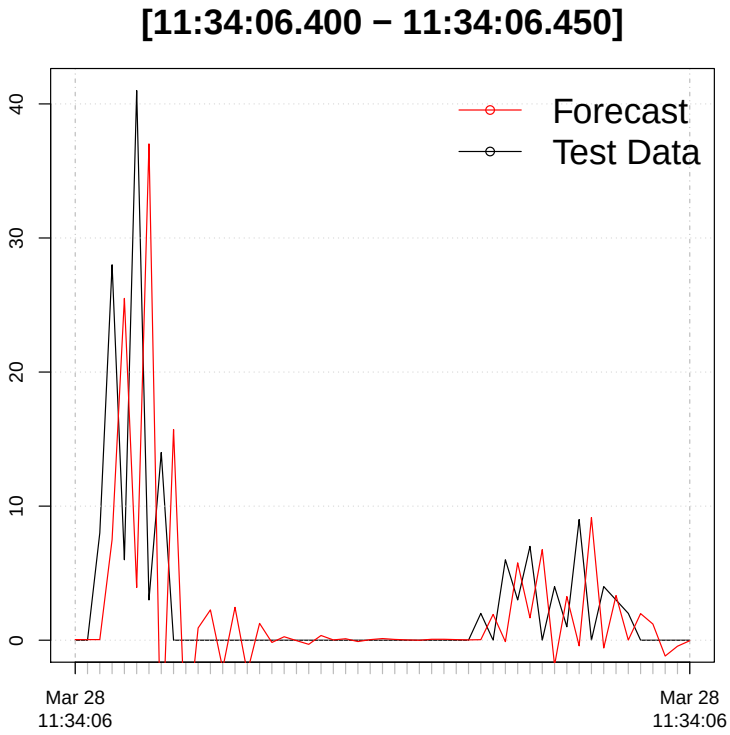


Figure 5.7: Time Series Plots on a Portion of the Test Data – ARIMA Model Developed for the Time Series of 1ms Intervals.

The ACF plot shows four spikes falling outside the recommended interval. This indicates that the residuals are correlated. The small value of the p in the portmanteau test (results shown in Table 5.1) also shows that the forecast can be improved by making use of information contained in the forecast errors. Consequently, the model will not produce optimal forecasts. This was anticipated, because the time series of 1ms intervals falls into a class of zero-inflated or integer valued time series data whose forecast models requires INteger valued AutoRegressive (INAR) models [116]. These models were not considered for this study, because the data was not known until the experiments were performed and this research specifically investigates the applicability of ARIMA and ANN as forecasting techniques for the problem in hand.

Table 5.2: Forecasts Accuracy Measures – ARIMA Model for 1ms Interval Time Series.

ME	**RMSE**	**MAE**	**MASE**
0.04154	2.6951	0.5549	1.3929

The accuracy measures of the forecasts are shown in Table 5.2. The MASE value which is more than one shows that the forecasts were obtained from a model that performs worse than a naive forecast model. The forecasts as well as the test data are shown in Figure 5.6. The highlighted window in Figure 5.6 is shown with more resolution in Figure 5.7. A more detailed view of the forecasts between [11:34:06.400 -to- 11:34:06.450] are shown in Figure 5.7. The plots show that the forecasts follows the average value of the original signal. This is a positive finding, because the average value can be used to indicate the average traffic intensity per unit time. Hence, this can be used to indicate the average resource requirements which will vary with time. However, using the developed forecasting model will be a waste of resources. One could just use the naive forecast and get more accurate forecasts.

5.6.1.2 ARIMA Forecasts of Traffic Intensity at 10ms Intervals

The time series of the traffic intensity sampled at 10ms is also characterized by many periods of zeros, therefore, the forecast model with ARIMA is expected to face similar problems as those encountered in the time series of 1ms intervals. The recordings were taken for 29.98 seconds [11:34:03.92 - 11:34:33.90], with 2999 records. The test and training sets were selected in a similar manner as those for the 1ms time series.

The model's evaluation by making use of residual analysis shows that the model for the time series of 10ms intervals performs slightly better than that of the time series of 1ms intervals. The ACF plot shown in Figure 5.8 has only two spikes that are outside the recommended interval. This means that the residuals are still correlated, but the correlation is not as strong as the one found in the time series of 1ms intervals. The portmanteau test results are given in Table 5.3. This also shows a small value of p, but larger than the one found in the ARIMA model developed for the time series of 1ms time intervals.

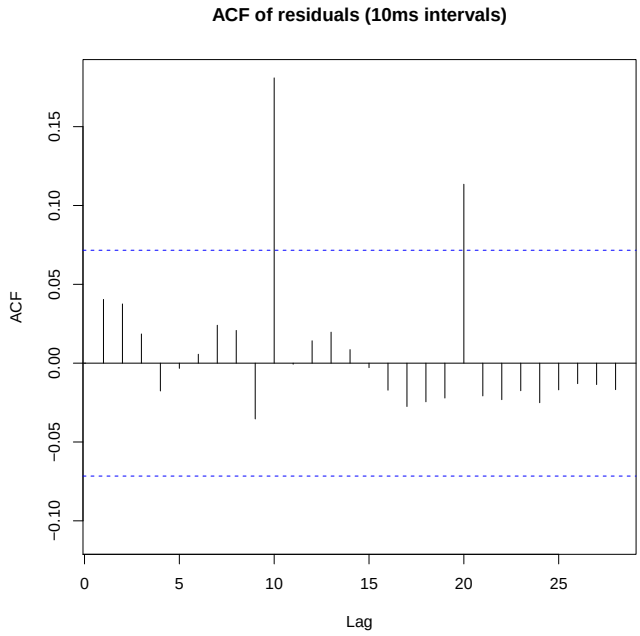


Figure 5.8: ACF Plot of Residuals – ARIMA Model Developed for the Time Series of 10ms Intervals.

Table 5.3: Results of a Portmanteau Test – ARIMA 10ms Time Series.

$h = 24$	$\chi^2 = 43.012$	$df = 24$	$p = 0.009915$

Due to the availability of correlation between the residuals and the low value of p obtained by the portmanteau test, the model will not provide optimal forecasts. Therefore, the model can still be improved to obtain better forecasts. This finding is also attributed to the above-mentioned concern that ARIMA models are not designed for low count time series. It is for this reason that attempts to improve the ARIMA model for this type of time series was not made.

Table 5.4 shows the accuracy measures. The MASE value shows that the model is still not better than a naive forecast model. Shown in Figures 5.9 and 5.10 are the time

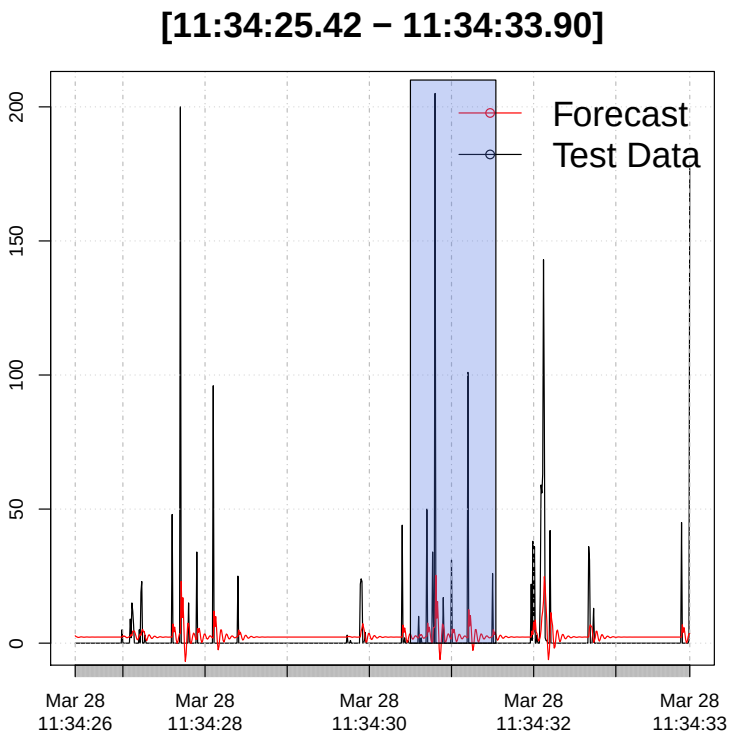


Figure 5.9: Time Series Plot of the Entire Test Data – ARIMA Model Developed for the Time Series of 10ms Intervals.

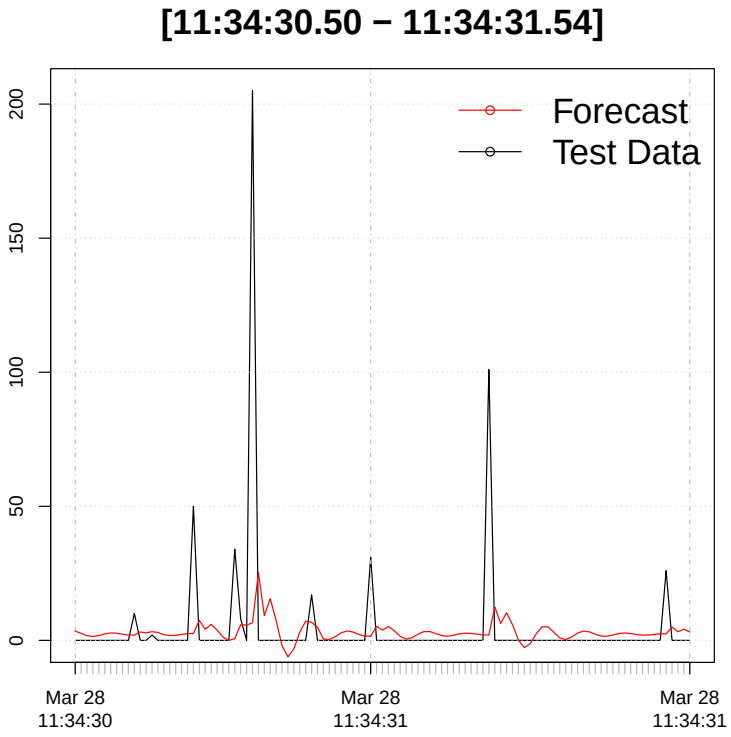


Figure 5.10: Time Series Plots on a Portion of the Test Data – ARIMA Model Developed for the Time Series of 10ms Intervals.

Table 5.4: Forecasts Accuracy Measures – ARIMA Model Developed for the Time Series of 10ms Intervals.

ME	RMSE	MAE	MASE
-0.05287844	15.61931	4.948515	1.237542

series plots of the forecasts and the test set. The time series of Figure 5.10 shows the interval highlighted in Figure 5.9 with a higher resolution. It is evident in Figure 5.10 that the forecasts only follow the average value of the predicted variable. Therefore, the ARIMA forecasts of the traffic intensity sampled at both 1ms and 10ms shows that ARIMA is not applicable to forecast the time series data recorded at these time intervals.

5.6.1.3 ARIMA Forecasts of Traffic Intensity at 1s Intervals

The time series plot of Figure 5.4 is continuously valued (i.e. assumptions made about the data in Section 5.6). It is in such a time series that ARIMA finds great applicability. The traffic intensity signal was recorded for a period of 1.067 minutes and 65 records were captured.

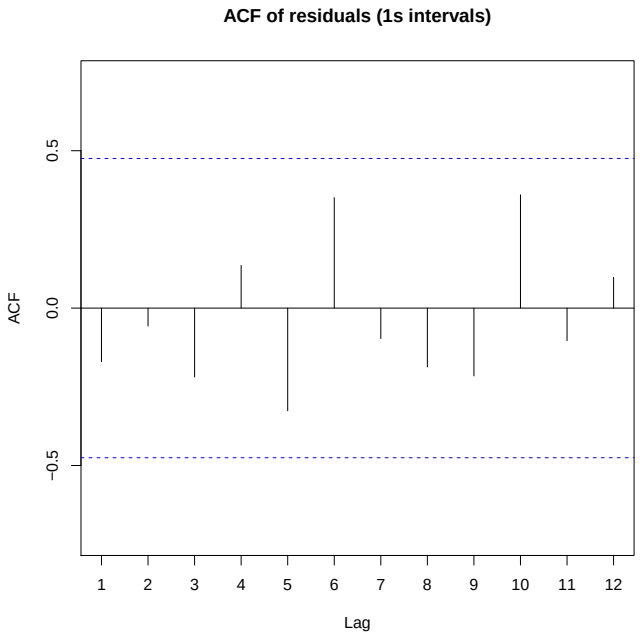


Figure 5.11: ACF Plot of Residuals – ARIMA Model Developed for the Time Series of 1s Intervals.

The training set for the model was chosen to be the first 48 records and the remaining

Table 5.5: Results of a Portmanteau Test – ARIMA 1s Time Series.

$h = 10$	$\chi^2 = 18.106$	$df = 10$	$p = 0.05231$

17 was chosen to be the test set. An ARIMA(4,0,0) model was developed. The portmanteau results and ACF plot are shown in Table 5.5 and Figure 5.11 respectively. The ACF plot shows all the spikes within the required interval, which entails that the residuals are not correlated. Therefore, an optimal forecast can be obtained from the developed model. The value of p from the portmanteau test is greater than 0.05 and it is more than five times larger than the one obtained previously. For this reason, the created model is considered acceptable and according to the ACF plot, it will produce an optimal forecast. The portmanteau test, also signals that small improvements can be made to the model to obtain better forecasts.

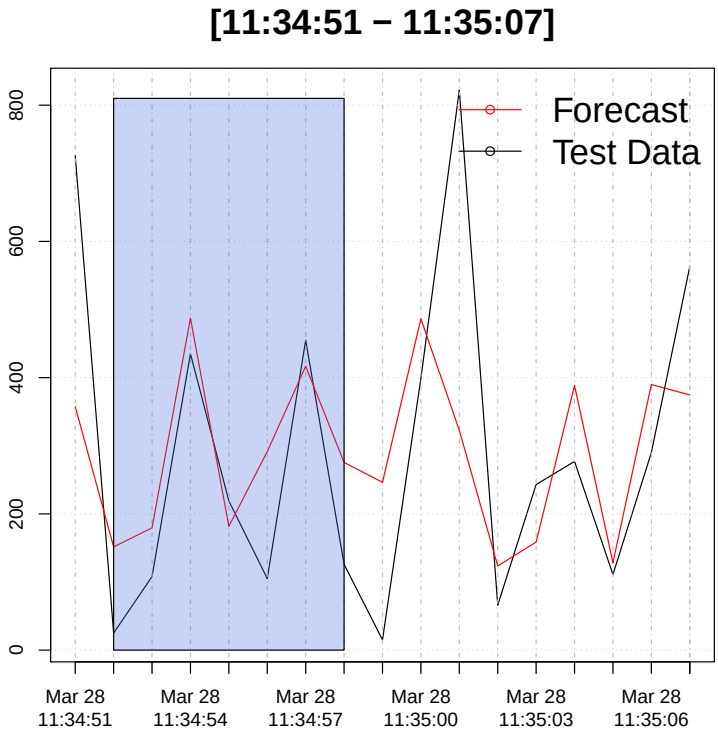


Figure 5.12: Time Series Plot on the Entire Test Data – ARIMA Model Developed for the Time Series of 1s Intervals.

Table 5.6: Forecasts Accuracy Measures – ARIMA Model Developed for the Time Series of 1s Intervals.

ME	**RMSE**	**MAE**	**MASE**
1.470186	187.6721	141.6305	0.490177

Accuracy measures from this model are given in Table 5.6. The interesting part of the accuracy measures is that the MASE value indicates that the results were obtained from a better forecasting model than a naive forecast model. Therefore, this indicates that ARIMA is applicable in forecasting this type of signal. Figure 5.12 shows the time series plot of the forecasts and the genuine time series data in the same plot. The highlighted region indicates that the forecasts are close to the real signal in parts where the signal does not have significant spikes. This means that for an unexpected rise in traffic the model is likely to give a wrong forecast value, but at the same time give a value close to an average.

Table 5.7: Results of a Portmanteau Test – ANN 1ms Time Series.

$h = 10$	$\chi^2 = 497.68$	$df = 10$	$p = 2.2e-16$

5.6.2 Artificial Neural Network (ANN) Models

Artificial Neural Networks (ANN) are used to approximate any class of functions without assuming any properties or behaviour of the data generating process. ANN models are expected to learn the behaviour of the process from the data. Hence, there is no need to check for conditions of stationarity in the time series as it was required in the ARIMA method.

5.6.2.1 ANN Forecasts at 1ms

The first time series data to be analyzed using the ANN model is the time series depicted in Figure 5.2, where the observations are recorded at 1ms intervals. The same data sets used in the ARIMA forecasts in Section 5.6.1.1 are used to train and test the ANN model. This is done to ensure that the forecasts obtained from both models (ARIMA and ANN) are kept in a similar scale to ensure that the values of the accuracy measures are comparable. In this case, forecast accuracy measures which are scale independent and those which are scale dependent can be used to compare the forecasts. The ANN model is developed by making use of the `nnetar` function which is part of the *forecast* package in *GNUR*. Like all statistical forecasting models, the technique of using ACF plots of the model's residuals and the portmanteau tests are

used to assess the model's ability to produce optimal forecasts.

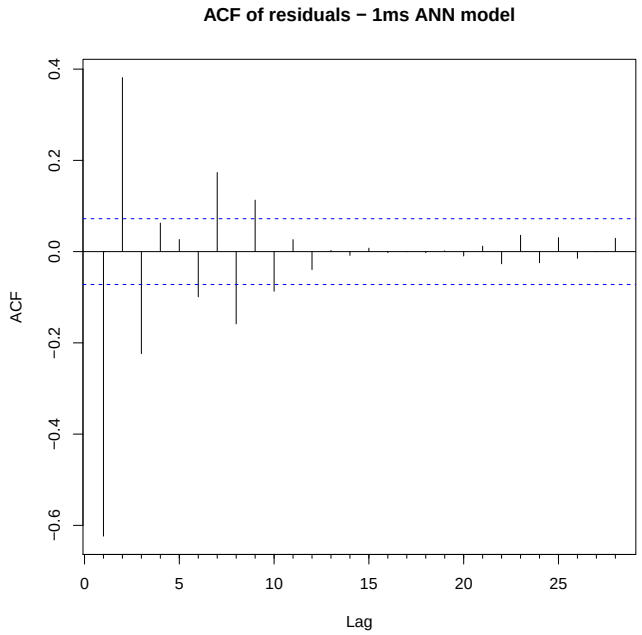


Figure 5.13: ACF Plot of Residuals – ANN Model Developed for the Time Series of 1ms Intervals.

An $ANN(10, 6)$ model was selected. In the notation $ANN(p, q)$, p represents the number of lagged values; q the number of hidden layer neurons and the ANN network consists of one output neuron whose output is not fed to an activation function. Table 5.7 shows the results of the portmanteau test. The value of p indicates that the residuals are not from a white noise signal. Shown in Figure 5.13 is the ACF plot of the residuals of the developed model. The ACF plot of the errors show that the model will not produce optimal forecasts. The spikes which goes beyond the recommended interval (at lags 1,2,3,6,7 and 8) indicate that the model does not account for the linear dependence at these lags. For this reason, the ANN model developed for the time series of 1ms intervals is not the optimal ANN model. The low count time series properties and the high number of zero values in the time series data are seen as the main cause for the failure of the ANN technique to come up with an optimal model.

Table 5.8: ANN Forecasts Accuracy Measures – ANN Model Developed for the Time Series of 1ms Intervals.

ME	**RMSE**	**MAE**	**MASE**
-0.0165	2.4331	0.4792	3.9048

The forecast accuracy measures are given in Table 5.8. This results show that the

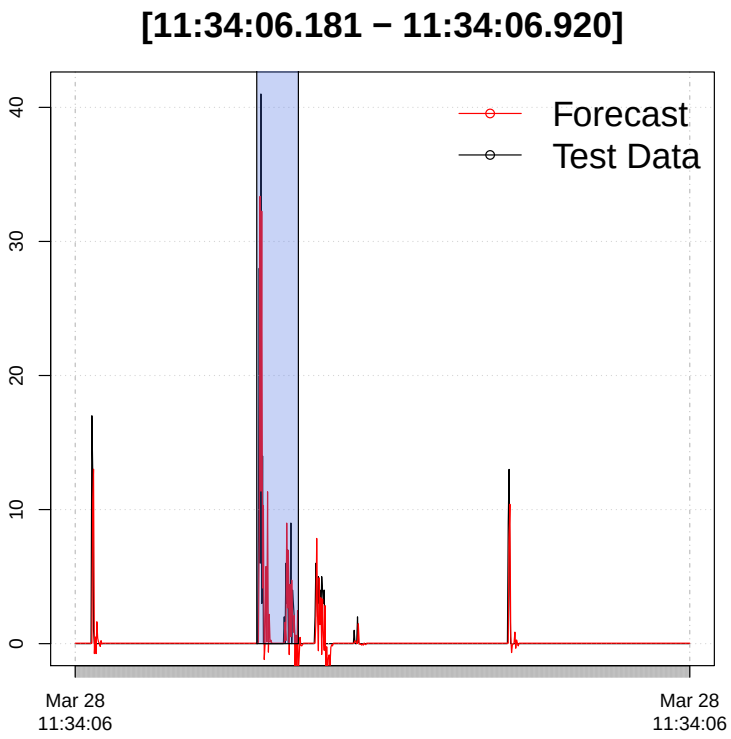


Figure 5.14: Time Series Plot of the Entire Test data – ANN Model Developed for the Time Series of 1ms Intervals.

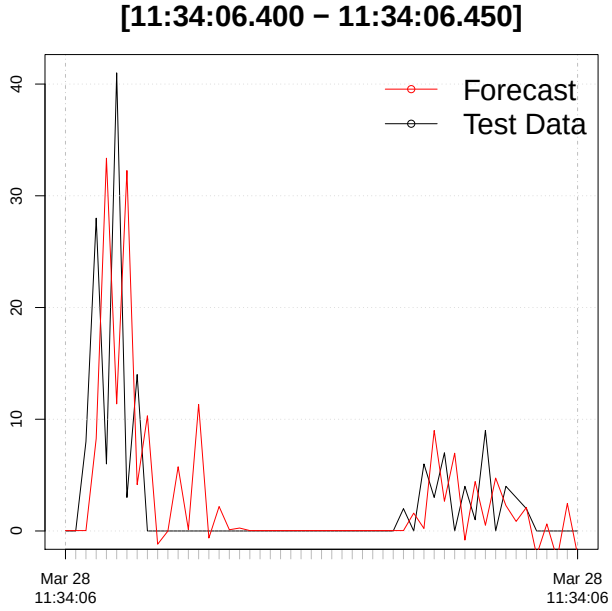


Figure 5.15: Time Series Plots on a Portion of the Test Data – ANN Model Developed for the Time Series of 1ms Intervals.

feed-forward ANN model developed performs poorly compared to the ARIMA method developed for the same data. The plot of the forecasts in the same set of axis as the true values is shown in Figure 5.14. The highlighted region in Figure 5.14 is shown in Figure 5.15 with more x-axis points.

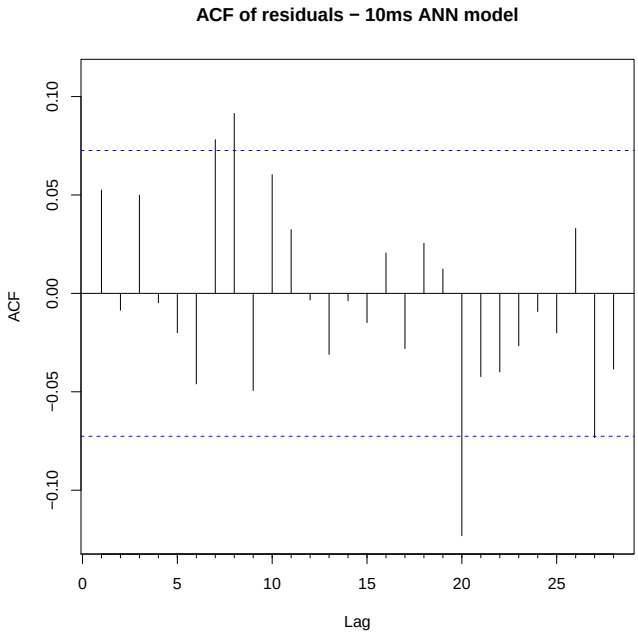


Figure 5.16: ACF Plot of Residuals – ANN Model Developed For the Time Series of 10ms Intervals.

Table 5.9: ANN Forecasts Accuracy Measures – ANN Model Developed for the Time Series of 10ms Intervals.

$h = 10$	$\chi^2 = 497.68$	$df = 10$	$p = 2.2e-16$

5.6.2.2 ANN Forecasts at 10ms

The time series in which the controller workload signal was sampled at 10ms interval is shown in Figure 5.3. The same training and test data sets as used in the ARIMA model were used to develop the ANN model. Evaluation of this model by making use of residual analysis indicate that the developed model will not provide optimal forecasts. However, it is important to note that the level of correlation in the time series of 10ms intervals is weaker than the one obtained in the time series of 1ms intervals. The ACF and portmantau results are shown in Figure 5.16 and Table 5.9. Both model evaluation techniques point to the same direction of poor forecasts.

Graphical plots of the test set and the forecasts time series are shown in Figures 5.17 and 5.18. Figure 5.18 shows an enlarged version of the highlighted region of Figure 5.17. It is evident in Figure 5.17 that the forecasts obtained from the ANN model are slightly better than those obtained in the ARIMA model for the same time series data. Forcast accuracy measures are contained in Table 5.10. These accuracy measures indicate that

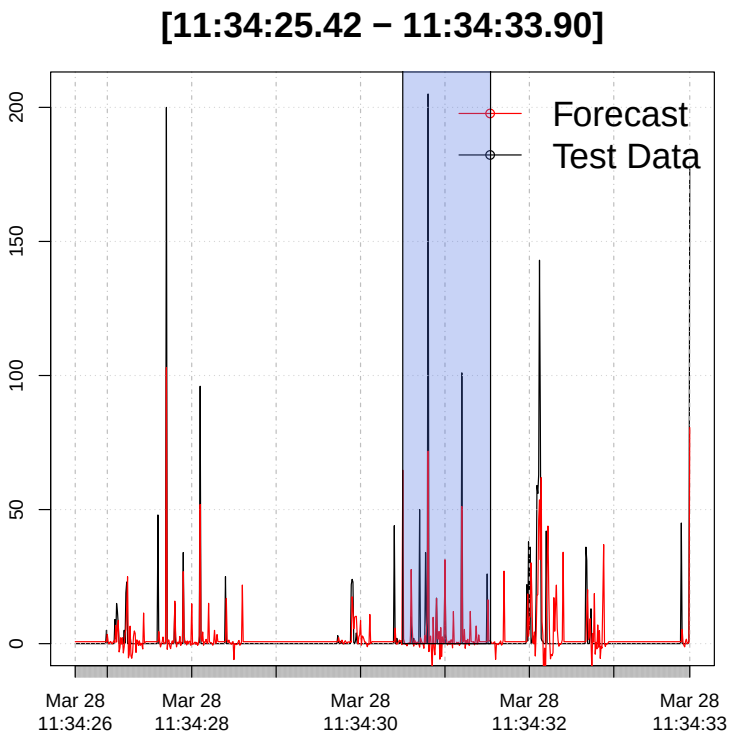


Figure 5.17: Time Series Plot on the Entire Test Data – ANN Model Developed for the Time Series of 10ms Intervals.

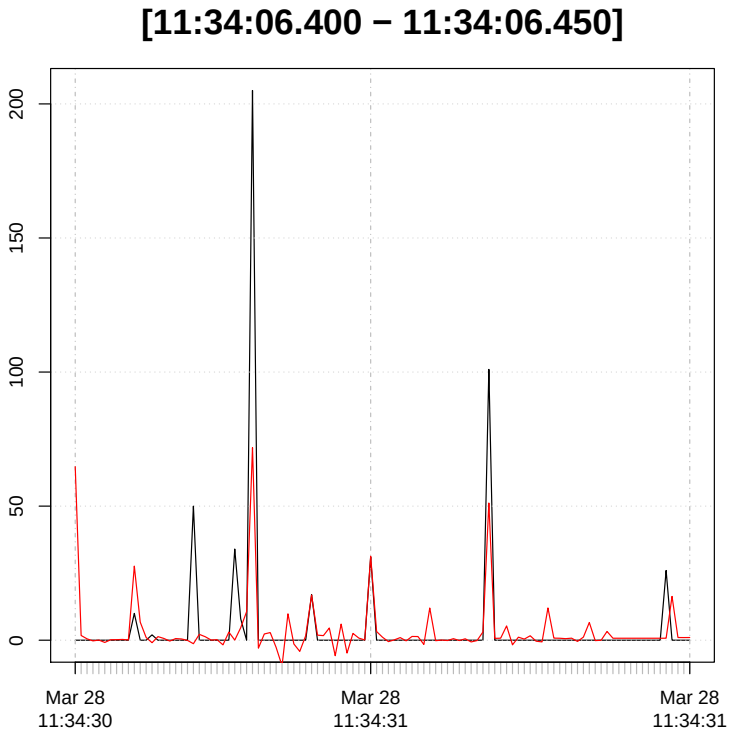


Figure 5.18: Time Series Plots on a Portion of the Test Data – ANN Model Developed for the Time Series of 10ms Intervals.

the forecasts obtainable from this model are better than those which could be obtained from a naive model or guessing. This is evident from the MASE value of 0.8706, which is less than one. Even though the ANN technique outperforms the ARIMA in this case, these results still show that both ARIMA and ANN are not good forecasting methods for time series signal which contain many periods of zeros. This is because the ACF plot (Figure 5.16) for this model indicated significant correlation between the residuals, which means that information contained in the errors could still be used to improve the model.

Table 5.10: ANN Forecasts Accuracy Measures – ANN Model Developed for the Time Series of 10ms Intervals.

ME	**RMSE**	**MAE**	**MASE**
-0.2195	11.4934	3.7242	0.8706

5.6.2.3 ANN Forecasts at 1s

The continuously valued time series of the controller workload variable sampled at 1s intervals is shown in Figure 5.4. It is in the forecasts of this time series that the forecasting capability of the ANN and ARIMA will be compared. This is because both techniques are regression based and are known to perform well on continuously valued time series signals. An ANN(4,2) model was developed for the time series of 1s intervals. This is a smaller ANN network compared to those developed for the low count time series of 1ms and 10ms intervals. It can also be stated that the smaller network signifies that the model developed for this time series is more robust compared to those developed in the previous two cases (i.e. the time series signals with 1ms and 10ms intervals).

Table 5.11: ANN Forecasts Accuracy Measures – ANN Model Developed for the Time Series of 1s Intervals.

$h = 10$	$\chi^2 = 8.0506$	$df = 10$	$p = 0.6239$

In efforts to evaluate the capability to provide optimal forecasts by the ANN(4,2) model, the residual analysis in the form of ACF plots and portmanteau test were used. The ACF plot and the portmanteau test results are shown in Figure 5.19 and Table 5.11. The ACF suggests that the residuals came from a white noise signal. This means

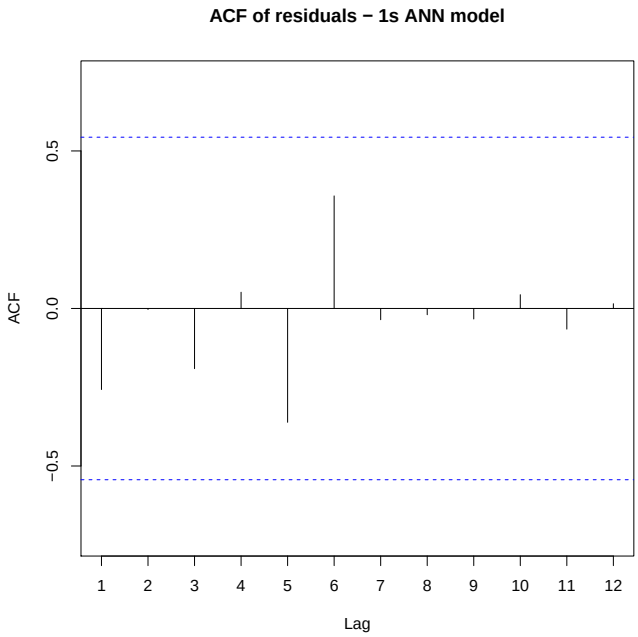


Figure 5.19: ACF Plot of Residuals – ANN Model Developed for the Time Series of 1s Intervals.

that there is no information in the errors which can be used to improve the forecasts. Therefore, the ACF plot entails that the developed ANN model is capable of producing optimal forecasts. The portmanteau test also gives a value of p that is big enough to suggests that the errors came from a white noise signal. The obtained value of p is larger than those obtained for the models developed for the time series signals of 1ms and 10ms intervals by a factor of 10^{15}. This means that this value is big enough for the portmanteau test to be considered to be in agreement with the ACF plot.

Table 5.12: ANN Forecasts Accuracy Measures – ANN Model Developed for the Time Series of 1s Intervals.

ME	**RMSE**	**MAE**	**MASE**
26.78172	189.995	146.6757	0.4913

Forecast accuracy measures are given in Table 5.12. The accuracy measure which is used to determine if the forecasts came from a forecast model that performs better than the naive forecast model is the MASE. This value is 0.4959, which suggests that the ANN model developed is better than guessing or the naive forecast model. The graph of the forecasts produced by the ANN is shown in Figure 5.20.

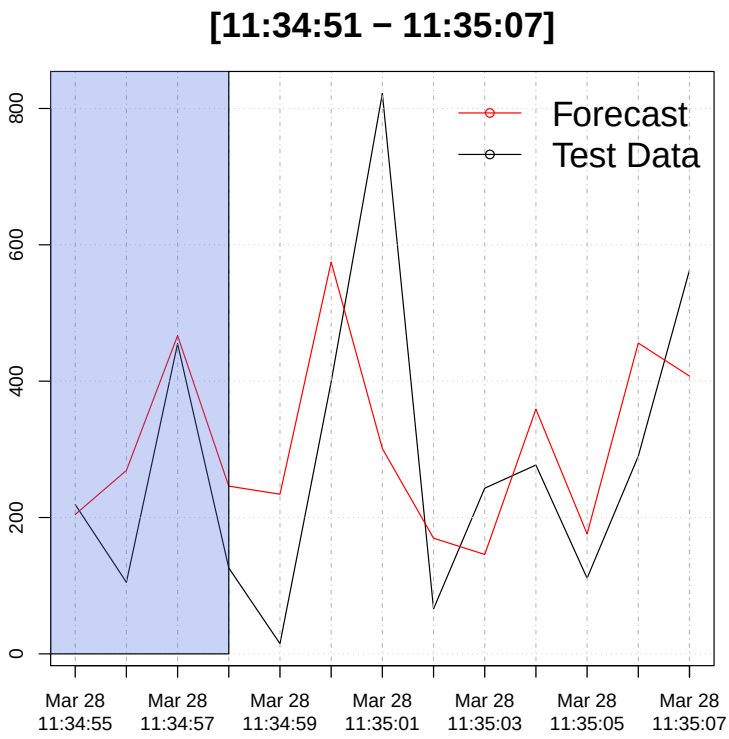


Figure 5.20: Time Series Plot on the Entire Test Data – ARIMA Model Developed for the Time Series of 1s Intervals.

5.7 Evaluation of Results

At the early stages of the research, the question answered in this chapter was originally stated as follows:

(A) *Which statistical models forecasts the network traffic reasonably well to perform load balancing tasks such as resource allocation at the controller plane? Resources in the controller plane are the required controller instances in distributed controllers and computing facilities such as the number of threads or cores in a multi-threaded controller.*

This is the original question of the research. The forecasting techniques selected to be investigated are ARIMA and ANN models. It was proposed that the answer be provided based on the following questions:

1. Can these methods forecast controller plane traffic?

2. In case these methods can provide forecasts, how do these methods perform when

compared to each other?

3. Are the forecasts applicable to controller plane load balancing? if yest to what extent?

4. In case these methods cannot forecast the traffic, why is it impossible and what other methods can be used to bring the traffic statistics into use in controller workload balancing methods?

The answer to question one is given in Section 5.6 and it is yes and no: Yes both ARIMA and ANN can forecast the controller plane workload when the time intervals of the time series are 1s wide. The answer is no for the time series signals with time intervals of 1ms and 10ms. The success of the forecasting models does not necessarily depend on the time intervals at which the controller workload variable is sampled, but on the values the time series variable assume at the time of the observation. It was discovered that if the time series is classified a low count or an intermittent time series, then both ARIMA and ANN produces poor forecasting models. This is consistent with many studies (referred to in Section 5.6) in forecasting of low count time series values. Therefore, it was realised that it is feasible to forecast the controller plane workload at any time interval with ARIMA and ANN, but the time series in question should not be zero-inflated or classifiable as a count time series data. This result also proves that the assumption that the zero-inflated time series encountered in this research belong to a group of count series is valid. This is indicated the development of poor forecast models by ARIMA and ANN, which are known to be applicable only in continuously valued time series data.

A benchmark which is used to test if it is feasible to forecast a given time series data set is the naive forecast model. It was discovered that the ANN and ARIMA models perform better that the naive forecast model for controller workload signal sampled at 1s intervals. The ANN was also found to be better than the naive forecast even for the time series of 10ms intervals. Furthermore, the ARIMA forecasting model is also applicable to determine the possibility of forecasting a given time series data [111]. This method also proves that forecasting of the controller workload signal is feasible, in the case of this research, the time series needs to satisfy the condition of being continuously valued.

Forecast models which are suitable for low count time series or the adaptation of the ANN or ARIMA to suit a specific case could have been sought. This was not done because the purpose and scope of this research was to determine the feasibility of using the ANN and ARIMA method for forecasting the controller workload. This research therefore opens a platform for the investigation of other forecasting models which can be used in the signals observed by SDN controllers.

Table 5.13: Comparisons of Accuracy the ANN and ARIMA Forecasts.

Forecast Model	**ME**	**RMSE**	**MAE**	**MASE**
ARIMA 1ms interval time series	0.0415	2.6951	0.5549	1.3929
ANN 1ms interval time series	-0.0165	2.4331	0.4792	3.9048
ARIMA 10ms interval time series	-0.0528	15.6193	4.9485	1.2375
ANN 10ms interval time series	-0.2195	11.4934	3.7242	0.8706
ARIMA 1s interval time series	1.47018	187.6721	141.6305	0.4902
ANN 1s interval time series	26.7817	189.9950	146.6757	0.4913

The second question above is about comparing these forcasting models against each other. Table 5.13 shows the accuracy measures obtained for both ARIMA and ANN. In some cases the ME (Mean Error) values are negative. This means that the forecasts were consistently below the true values. The same accuracy information is contained in the RMSE and MAE, which are positively valued. For the time series of 1ms and 10ms intervals, the ANN model outperforms the ARIMA. However, these models could not produce optimal forecasts for the time series of 1ms and 10ms intervals. For the time series of 10ms intervals, the ANN is more accurate compared to the ARIMA and according to the MASE it is better than guessing or the naive forecast. It was also noted that with increasing the number of iteration for the back-propagation algorithm of the ANN, more accurate forecasts could be obtained. This approach was not followed, because it could result in an over-fitted model and such models are not recommended in forecasting [100].

The assumption that the time series of 1s intervals is continuously valued was found to be valid. This is because both ARIMA and ANN which are widely used for forcasting continuously valued time series data were found to be capable of producing optimal forecasts for this time series (i.e. the time series of 1s time intervals). The ARIMA

model developed is found to perform better than the ANN. The difference in the accuracy measures is small – 0.001 in the MASE. This means that the performance of these models is more or less the same. Since both models produced residuals which are uncorrelated and normally distributed, the 95% prediction intervals were calculated: For the ANN it was found to be $\hat{y}_t \pm 22.798$ and for the ARIMA it was found to be $\hat{y}_t \pm 21.743$ ($\hat{y}_t$ is the forecast of y_t). Essentially, these prediction intervals also indicate that the two models perform almost similarly. The ARIMA is found to be favoured because it does not require as much computation as the ANN. On the other hand, the ANN may be favoured because it does not presume any properties of the data generation process and it can take into account non-linearities in the data. This means that the nature of the traffic has to be analysed before a suitable forecasting method is chosen.

The third question listed above is aimed at determining if forecasting the controller workload is applicable in alleviation of the possible controller plane congestion. It has been mentioned in the previous chapters (Chapter 2) that the scalability of the controller plane is determined by the information contained in the network-wide view as well as the mechanisms through which network control applications access this information. At the current state of the controller planes, the network-wide view represent the state of the data plane elements and *not* that of the controller plane. Therefore, the forecasts are aimed at creating a representation or an abstraction of the state of the controller plane. This means that control and management applications running on top of the controller plane can use this information in making network decisions. Forecasts are not accurate, but give an indication of the most likely values of a random quantity.

The use of the forecasts by a network application may involve the application determining a threshold value above which the controller plane is said to be congested. In case the prediction interval of the forecasts consistently include the threshold value, the network application may then decide to increase controller plane resources such as new controller nodes, requests more processing threads and others. The Onos controller used in this research takes about 40ms to discover the topology of a network. The same controller targets about 50ms for a device (e.g. a switch) to be discovered or for it to be registered in the controller's network-wide view [119]. A device with this capability can assume the control of about 10 switches within a period of 0.5s. It

could take the Onos controller about 0.8s to discover the 16 hosts used in the emulated network. Therefore, this means that if one forecasts at intervals of 1s and discover the need for a new device within the next 1s, then the control application will have enough time to get the required controller instance online. It is important to point out that the controller does not have to discover the existing network-wide view by detecting all network devices, this information could be accessed easily from the existing databases like the Network Information Base (NIB) used in the Onix controller platform.

The results of this research leaves the implementation of network applications that uses the forecast as a task for network application developers. It is in such applications that the correct forecast horizon will be determined. This forecast horizon should take into account the network size and the latency associated with the chosen controller. For the use of the ARIMA or ANN (i.e. the feed-forward), the forecaster will have to ensure that the signal is continuously valued at the time intervals at which is observed. The intervals at which observations are made are also critical because the flow statistics are hosted in the switches, so the application developer should ensure that requests for information from the switches should not increase the network's overhead.

5.8 Chapter Summary

A brief introduction of time series analysis, ARIMA and ANN is given in this chapter. ANN and ARIMA models were developed to forecast the controller workload signal which was sampled at 1ms, 10ms and 1s. Zero-inflated time series could not find optimal forecasting models on both the ARIMA and ANN models considered. These time series data was encountered where the controller workload variable was sampled at time intervals of 1ms and 10ms. The time series of 1s time intervals was found to be continuously valued, which made it possible for ARIMA and ANN to be applicable in forecasting its future values. The use of the naive forecast as a benchmark which a forecast model should beat before it is considered a usable forecast model showed that both ARIMA and ANN are usable in forecasting a continuously valued controller workload time series data. Although the ANN was found to perform better than the naive forecast model in the intermittent time series of 10ms intervals, it was found that the development of such a model required a large number of iterations of the back-propagation algorithm. Moreover, the forecasts were also not as accurate as those

obtained in the time series of the time series data observed at 1s intervals. The obtained results indicate that forecasting approaches are applicable to forecast the controller-plane workload variable. The usability of the forecasts was tested by considering the performance measures of an ONOS controller platform. The results showed that a forecast horizon of 1s is large enough for an ONOS controller to load a new controller node in a distributed controller plane. Thus, the capability to forecast the workload at 1s is considered a positive finding, because the forecast horizon enables controller plane to successfully react to information provided by the forecasts.

6 CONCLUSION

The objective of this research was to answer two primary questions: The first question was aimed at revealing the statistical properties of the traffic seen by SDN controllers. The second question was aimed at determining the feasibility of forecasting the traffic intensity experienced by SDN controllers. It is believed that answers to the first question will play a vital role in the evaluation of SDN controllers (during controller design stages) and answers to the second question will help in alleviation of controller plane congestion and therefore help in the designs of scalable SDN controllers.

This chapter is structured as follows: Section 6.1 presents the contribution of this research work. Section 6.2 discusses possible future research work which can be used to explore more features of the controller plane workload variable.

6.1 Research Contribution

The first outcome of this research is the characterisation of the controller workload variable in terms of its statistical properties. In the past, this arrival process was assumed to follow Markov models such as the M/M/1 queueing system. These models assume a Poison arrival process where the inter-arrival times are exponentially distributed. It is known that these models have been widely studied and even taken as a convention in arrival processes. This research points out that SDN controllers see flows with inter-arrival times that can be best modelled by a beta distribution and not an exponentially distributed random variable. This finding is considered novel for SDN controllers, but consistent with inter-arrival times reported for Machine Type Communications (MTC) in studies like [25], [97] and [96]. Therefore, the arrival process of flows seen by an SDN controller is less random compared to other arrival processes. This comes from the property that beta distributions have a finite support range and are therefore applicable in the description of random variables that are associated with finite time lengths [97]. This means that the inter-arrival times associated with flow

initiations at the controller plane are of finite length. It is important to note that this is an important feature in search engines (or other data center applications), where flows to search for results from other servers are expected to return results within a specified period. Hence, the inter-arrival times of some of these flows are deterministic or have an initial description which can be used to provide additional information about future inter-arrival times.

The distribution of the arrival times show that both tails of the distribution are important in the evaluation of controller plane performances. The left tail represents small inter-arrival times which are associated with concurrent flows or those linkable to applications whose packets have deadlines like those of search engines. The right tail represent flows with longer inter-arrival times, which can be flows that come after a network failure event. Such inter-arrival times are also important for the investigation of latencies associated with network recovery or the ability of the controller plane to recover from failure. It is believed that the information on the distribution of arrival times will play a vita role in the evaluation of newly designed SDN controllers and control applications. At the writing of this research report, there were not studies which had discovered that the inter-arrival times of flows seen by SDN controllers can be modelled by a beta distribution and that both tails of the distribution are important for the evaluation of performance of SDN controllers. It was also found that 1-2% of flows arrive within 10μs of each other while more than 80% have inter-arrival times in the range of 10μs–10ms. This is consistent with flow characteristics observed in published data center studies.

Traffic intensity forecasts are the second outcome from this research. This is intended at designing controller planes that are aware of the workload they carry and that which they can anticipate in the future. Essentially these are controller planes which are aware of possible congestion or possible degradation of service offered by applications running in the network. For example, consider a delay sensitive application such as a voice transmitting application (e.g.VoIP). Packets for such an application will normally be given priority at switching devices and at the controller plane. However, in a situation where the controller plane has all its resources occupied, such packets are likely be enqueued until controller plane resources become available. A controller plane which is aware of its current utilization will be able to detect if it will no longer be able to handle priority flow initiations such as those of voice packets. It is for this reason

that this research proposes a representation of the controller plane resource usage at the network-wide view of the network which is kept by the controller plane to realize a logically centralized controller platform. This research proposes to represent the utilization of the controller plane in the network-wide view by its forecasts.

The forecasts are applicable in this case because they are created from the history of the traffic intensity. Thus, there is no need for direct measurement techniques which have high network overhead. The history of the flows can be requested at varying time intervals from the switches. Moreover, the amount of history required depends on application requirements. For example, the ARIMA model developed for the time series data observed at intervals of 1s intervals required only the past four values of the traffic history to produce an optimal forecast. This means that applications could determine the level of accuracy required for their purposes and therefore develop forecast models that meet the required accuracy.

This research did not develop applications that utilize forecast to test improvements on controller plane platforms. The aim of the research was only to deterimne the feasibility of using forecast to allocate controller plane resources. The result show that it is possible to use forecasts to allocate controller plane resources. The requirement for useful forecasts when using ANN or ARIMA is a continuously valued time series. The forecast horizon is determined by the resources to be allocated. For example, this research showed that with a forecast horizon of 1s, it is possible to add an ONOS controller to a network and get it to assume the control about 10 switches within half a second. The allocation of threads can be done in order of microseconds depending on the computing resources available. Therefore, this research opens a platform for SDN application developers to delve into other forecasting techniques which can see the controller workload predicted with more accuracy. These initiatives will play a role in allevation of controller plane congestions or degraded quality of service in delay sensitive applications such as video, voice and mobility (in mobile network hand-overs).

The research work carried out is therefore considered to have accomplished its objectives. The statistical properties of the controller workload variable were determined and the feasibility of employing statistical forecasting techniques to the controller workload was established. It was also discovered that the forecasts are applicable in the alleviation of controller plane congestion. It is believed that the use of appropriate traffic

patterns for the evaluation of performance of SDN controllers will improve the standards of testing SDN controllers and consequently help in the design of more effective controller planes and SDN-based networks.

6.2 Future Work

This research work used a data center network to study properties of the traffic observed by an SDN controller. It is not known if the same patterns will be observed in other networks such as WANs, LANs and others. Therefore, some research work may have to be done on other networks. The size of the network used in this study is small compared to real life systems. This was due to shortage of computing resources. However, the scale of the network was not considered important in this research. It will be useful to test the results of this research on a network much bigger than the one used.

The forecasts obtained in this study can be improved by combining ARIMA and ANN to create a better forecasting models. Other studies have demonstrated that a combination of the two methods provide more accurate forecasts. This could be done in studies where the accuracy of the forecasts is a priority. Furthermore, as it was discovered that the inter-arrival times of flows in the controller plane is less random and knowledge about these flows is linked to applications running in the network. It is therefore necessary for researchers to improve the forecasts by making use of techniques from inference or machine learning. For example, the outcome of a machine learning algorithm which detects the type of application generating the flows can be used as an input to a forecasting algorithm. The forecast model will then take into account the inter-arrival times associated with that application to come up with a more accurate forecast of the expected traffic intensity. It could also be useful to investigate the feasibility of detecting an application by making use of the flows it generates. Such an application could find great applicability in tasks where communications service providers need to monitor the usage of certain applications.

Bibliography

[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, pp. 14–76, January 2015.

[2] Open Networking Foundation, *SDN Architecture*. Palo Alto, CA, USA, issue 1 ed., June 2014. ONF TR-502.

[3] Open Networking Foundation, *OpenFlow Switch Specification*, version 1.0.0 (wire protocol 0x01) ed., December 31 2009. ONF TS-001.

[4] W. Braun and M. Menth, "Software-Defined Networking Using Openflow: Protocols, Applications and Architectural Design Choices," *Future Internet*, vol. 6, pp. 302–336, May 2014.

[5] Open Networking Foundation, *OpenFlow Switch Specification*, version 1.5.1 (wire protocol version 0x06) ed., March 26 2015. ONF TS-025.

[6] A. Greenberg, G. Hjalmtysson, D. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A A Clean Slate 4D Approach to Network Control and Management," in *ACM SIGCOMM Computer Communication Review*, 2005.

[7] J. Sahoo, S. Mahopatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pp. 222–226, IEEE, April 2010.

[8] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, pp. 123–137, Aug. 2015.

[9] M. Khashei and M. Bijari, "An artificial neural network (p,d,q) model for time-series forecasting," *Expert Syst. Appl.*, vol. 37, pp. 479–489, Jan. 2010.

[10] Gartner, "IT Glossary." `http://blogs.gartner.com/it-glossary/page/2/?s=Network`. Last accessed: 05-04-2016.

[11] MICROFOCUS, "Chapter One: Security Overview." `http://support.novell.com/techcenter/articles/ana19940401.html`. Last accessed: 05-04-2016.

[12] 3GPP, "IP Multimedia subsystem." `http://www.3gpp.org/technologies/keywords-acronyms/109-ims`. Last accessed: 06-04-2016.

[13] ITU, "ITU-T's Defination of NGN." `http://www.itu.int/en/ITU-T/gsi/ngn/Pages/defination.aspx`. Last accessed: 06-04-2016.

[14] OECD (2015), "Triple and Quadruple Play Bundles of Communication Services," *OECD Science, Technology and Industry Ploicy Papers*, no. 23, 2015. OECD Publishing, Paris. `http://dx.doi.org/10.1787/5js04dp2q1jc-en`.

[15] L. Cui, F. Yu, and Q. Yan, "When Big Data Meets Software-Defined Networking: SDN for Big Data and Big Data for SDN," *IEEE Networks*, January 2016.

[16] B. Marr, "Why only one of the 5 Vs of big data really matters." `http://www.ibmbigdatahub.com/blog/why-only-one-5-vs-big-data-really-matters`. Last accessed: 06-04-2016.

[17] Open Networking Foundation, "Software-defined networking: The New Norm for Networks," *ONF White Paper*, 2012
.

[18] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–98, Apr. 2014.

[19] Open Network Foundation, "ONF Overview." `https://www.opennetworking.org/about/onf-overview`. Last accessed: 16-04-2016.

[20] Linux Foundation Collaborative Projects: OPENDAYLIGHT, "OpenDaylight Platform." `https://www.opendaylight.org/`. Last accessed: 16-04-2016.

[21] Open Network Foundation, "Technical library." `https://www.opennetworking.org/sdn-resources/technical-library`. Last accessed: 24-04-2016.

[22] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chuo, "A raodmap for traffic engineering in SDN-OpenFlow networks," *Computer Networks*, vol. 71, pp. 1–30, June 2014.

[23] Mininet, "Mininet An Instant Virtual Network on you Laptop (or other PC)." `http://www.mininet.org/overview`.
Last Accessed: 05-04-2016.

[24] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "Onos: Towards an open, distributed sdn os," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, (New York, NY, USA), pp. 1–6, ACM, 2014.

[25] G. TR37.868, "Study on ran improvements for machine-type communications;(release 11)," tech. rep., 3GPP, September 2011.

[26] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 92–99, Jan. 2010.

[27] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pp. 267–280, ACM, 2010.

[28] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, (New York, NY, USA), pp. 202–208, ACM, 2009.

[29] Z. K. Khattak, M. Awais, and A. Iqbal, "Performance evaluation of opendaylight sdn controller," in *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 671–676, December 2014.

[30] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an Operating System for Networks," in *In SIGCOMM CCR*, July 2008.

[31] E. Ng, Z. Cai, and A. Cox, "Maestro: A System for Scalable OpenFlow Control," tech. rep., Department of Computer Science: Rice University, 2010.

[32] D. Erickson, "The beacon openflow controller," tech. rep., Stanford University, 2010.

[33] Project Floodlight, "Floodlight Is an Open SDN Controller." `http://www.projectfloodlight.org/floodlight/`. Last accessed: 10-05-2016.

[34] Project FloodLight, "Module Descriptions and Javadoc." `https://floodlight.atlassian.net/wiki/display/floodlightcontroller/`. Last accessed: 10-05-2016.

[35] big switch networks, "Contact Big Switch SDN Experts." `http://bigswitch.com/company/contact`. Last accessed: 10-05-2016.

[36] trema, "Trema Full-Stack OpenFlow Framework in Ruby and C." `https://trema.github.io/trema/`. Last accessed: 10-05-2016.

[37] NEC Corporation of America 2016, "Smart tranformations." lhttps://www.necam.com/. Last accessed: 13-05-2016.

[38] NEC Corporation 2012 Thomas Dietz , "Trema Tutorial." `http://www.fp7-ofelia.eu/assets/Uploads/201203xx-TremaTutorial.pdf`. Last accessed: 10-05-2016.

[39] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, (Berkeley, CA, USA), pp. 351–364, USENIX Association, 2010.

[40] The Apache Software Foundation , "Cassandra." `http://cassandra.apache.org/`. Last accessed: 14-05-2016.

[41] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, (Berkeley, CA, USA), pp. 3–3, USENIX Association, 2010.

[42] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 19–24, ACM, 2012.

[43] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," in *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, (New York, NY, USA), pp. 351–362, ACM, 2010.

[44] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, (New York, NY, USA), pp. 254–265, ACM, 2011.

[45] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, (Berkeley, CA), USENIX, 2012.

[46] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–9, May 2014.

[47] Cisco, "Cisco IOS NetFlow." `http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html`. Last accessed: 15-05-2016.

[48] Cisco, "Introduction to cisco ios netflow," tech. rep., Cisco, May 2012.

[49] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca, "Planck: Millisecond-scale monitoring and control for commodity networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 407–418, Aug. 2014.

[50] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "Opentm: Traffic matrix estimation for openflow networks," pp. 201–210, PAM, 2010.

[51] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, (New York, NY, USA), pp. 8:1–8:12, ACM, 2011.

[52] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in *Proceedings of the 14th International Conference on Passive and Active Measurement*, PAM'13, pp. 31–41, Springer-Verlag, 2013.

[53] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pp. 29–42, USENIX Association, 2013.

[54] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for commodity sdn," in *Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems*, ICDCS '14, pp. 228–237, IEEE Computer Society, 2014.

[55] H. Nikhil, S. Seetharaman, M. Flajslik, A. Gember, N. Mckeown, G. Parulkar, A. Akella, N. Feamster, R. Clark, A. Krishnamurthy, V. Brajkovic, T. Anderson, D. T. R, and D. L. Usa, "Aster*x: Load-balancing web traffic over wide-area networks," 2009.

[56] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 7–12, ACM, 2012.

[57] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Scotch: Elastically scaling up sdn control-plane using vswitch based overlay," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, (New York, NY, USA), pp. 403–414, ACM, 2014.

[58] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Elasticon: An elastic distributed SDN controller," in *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '14, (New York, NY, USA), pp. 17–28, ACM, 2014.

[59] B. Heller, *Reproducible Network Research with high-Fidelity Emulation*. PhD thesis, Stanford University, June 2013.

[60] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *HotNets*, no. 10, pp. 19:1–19:6, ACM, 2012.

[61] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *CoNEXT*, pp. 253–264, ACM, 2012.

[62] S. Khan, B. Aziz, S. Najeeb, A. Ahmed, M. Usman, and S. Ullah, "Reliability of network simulators and simulation based research," in *2013 IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 180–185, Sept 2013.

[63] DRPA, "The Network Simulator." `http://www.isi.edu/nsnam/ns`. Last Accessed: 05-04-2016.

[64] Opnet, "Opnet Modeller."
`http://www.riverbed.com/za/products/steelcentral/opnet.html`.
Last Accessed: 05-04-2016.

[65] SCALABLE NETWORK TECHNOLOGIES, "QualNet The fastest most scalable network modelling platform."
`http://web.scalable-networks.com/content/qualnet`.
Last Accessed:24-05-2016.

[66] School of Computer Science University of Utah, "Emulab - Network Emulation Testbed." `http://www.emulab.net`.
Last accessed: 05-04-2016.

[67] GENI, "GENI Network Innovators Community Event." `http://www.geni.net`. Last Accessed: 05-04-2016.

[68] OFELIA, "Openflow in Europe: Linking Infrastructure and Applications." `http://www.fp7-ofelia.eu`. Last Accessed: 05-04-2016.

[69] S. Luo, Z. Lin, and X. Chen, "Virtualization security for cloud computing service," in *Cloud and Service Computing (CSC), 2011 International Conference on*, pp. 174–179, IEEE, December 2011.

[70] S. Luo, Z. Lin, and X. Chen, "Virtualization technology and its impact on computer hardware architecture," in *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, pp. 719–724, IEEE, April 2011.

[71] A. Joy, "Performance comparison between linux containers and virtual machines," in *Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in*, pp. 342–346, IEEE, March 2015.

[72] O. vSwitch, "Production quality, multilayer open virtual switch." `http://openvswitch.org/`.
Last Accessed: 08-08-2016.

[73] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 63–74, Aug. 2008.

[74] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, pp. 183–197, Aug. 2015.

[75] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, (Berkeley, CA, USA), pp. 19–19, 2010.

[76] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, (Berkeley, CA, USA), pp. 19–19, USENIX Association, 2012.

[77] W. Wang, Y. Sun, K. Salamatian, and Z. Li, "Adaptive path isolation for elephant and mice flows by exploiting path diversity in datacenters," *IEEE Transactions on Network and Service Management*, vol. 13, pp. 5–18, March 2016.

[78] AppNeta and F. Klassen, "Tcpreplay." `http://tcpreplay.appneta.com/`. Last accessed: 31-05-2016.

[79] A. Botta, A. Dainotti, and A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.

[80] P. Wette and H. Karl, "Dct2gen," *Comput. Commun.*, vol. 80, pp. 45–58, Apr. 2016.

[81] P. Wette, "DCT2Gen: A traffic Genarator for Data Centers." `http//:www-old.cs.uni-paderborn.de/en/research-group/research-group-computer-networks/people/dr-philip-wette/dct2gen.html`. Last accessed: 31-05-2016.

[82] B. Lantz, N. Handigol, B. Heller, and V. Jeyakumar, "Introduction to mininet." `https://github.com/mininet/mininet/wiki/Introduction-to-Mininet`. Last accessed: 22-08-2016.

[83] Mininet, "Mininet python api reference manual." `http://mininet.org/api/annotated.html`. Last accessed: 22-08-2016.

[84] J. Noller, "Ssh programming with paramiko | completely different." `http://jessenoller.com/blog/2009/02/05/ssh-programming-with-paramiko-completely-different`. Last accessed: 22-08-2016.

[85] OpenSSH, "Openbsd ssh." `http://man.openbsd.org/OpenBSD-current/man8/sshd.8`. Last accessed: 22-08-2016.

[86] Wireshark, "About Wireshark." `http://www.wireshark.org`. Last Accessed: 04-05-2016.

[87] *Handbook, TELETRAFFIC ENGINEERING*. Geniva: International Telecommunication Union and the International Teletrafic Congress, December 2003.

[88] M. Delignette-Muller and C. Dutang, "fitdistrplus: An R package for fitting distributions," *Journal of Statistical Software*, vol. 64, February 2015.

[89] A. Cullen and H. Frey, *Probabilistic Techniques in Exposure Assessment.* Plen Publishing Co., 1st ed., 1999.

[90] M. Gupta, J. Sommers, and P. Barford, "Fast, accurate simulation for sdn prototyping," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, (New York, NY, USA), pp. 31–36, ACM, 2013.

[91] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," *SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. 350–361, Aug. 2011.

[92] MathWorks, "MATLAB: The Language of Technical Computing." `http://www.mathworks.com/products/matlab/`. Last accessed: 01/08/2016.

[93] The R Project for Statistical Computing, "Getting Started." `http://www.r-project.org`. Last accessed: 05-04-2016.

[94] A. Feldmann and W. Whitt, "Fitting mixtures of exponentials to long-tail distributions to analyze network performance models," in *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, vol. 3, pp. 1096–1104 vol.3, Apr 1997.

[95] J. P. Nolan, "Fitting data and assessing goodness-of-fit with stable distributions," tech. rep., Department of Mathematics and Statistics American University, Washington, DC 20016 USA, June 1999.

[96] X. Jian, X. Zeng, Y. Jia, L. Zhang, and Y. He, "Beta/m/1 model for machine type communication," *IEEE Communications Letters*, vol. 17, pp. 584–587, March 2013.

[97] X. Jian, X. Zeng, Y. Jia, J. Haung, and Y. Zhou, "Statistical description and analysis of the concurrent data transmission from massive mtc devices," *International Journal of Smart Home*, vol. 8, pp. 139–150, July 2014.

[98] R. H. Shumway and D. S. Stoffer, *Time Series Analysis and Its Applications (Springer Texts in Statistics).* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[99] J. D. Cryer and K.-S. Chan, *Time Series Analysis with Applications in R (Springer Texts in Statistics)*. 233 Spring Street, New York, NY 10013, USA: Springer-Verlag New York, Inc., 2008.

[100] R. J. Hyndman and G. Athanasopoulos, *Forecasting Principles and Practice*. OTEXT.COM, 2014.

[101] E. Stellwagen and L. Tashman, "Arima: The model of box and jenkins," *FORESIGHT: The International Journal of Applied Forecasting*, no. 30, pp. 29–33, 2013.

[102] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 233 Spring Street, New York, NY 10013, USA: Springer, second ed., Aug. 2009.

[103] F. rodrigues, C. Cardeira, and J. Calado, "The daily and hourly energy consumption and load forecasting using artificial neural network method: A case study using a set of 93 households in portugal," *Energy Procedia*, vol. 62, pp. 220–229, December 2014.

[104] T. M. Mitchel, *Machine Learning*. McGraw-Hill Science/Engineering/Math, March 1997.

[105] G. Ljung and G. Box, "On a measure of lack of fit in time series models," *Biometrika*, vol. 65, 1978.

[106] R. Hyndman, "Thoughts on the Ljung-Box test." `http://www.robjhyndman.com/hyndsight/ljung-box-test`. Last accessed: 19-09-2016.

[107] C. Stolojescu-Crisan, "Data mining based wireless network traffic forecasting," in *Electronics and Telecommunications (ISETC), 2012 10th International Symposium on*, pp. 115–118, Nov 2012.

[108] P. Cortez, M. Rio, M. Rocha, and P. Sousa, "Internet traffic forecasting using neural networks," in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pp. 2635–2642, 2006.

[109] C. Katris and S. Daskalaki, "Comparing forecasting approaches for internet traffic," *Expert Systems with Applications*, vol. 42, no. 21, pp. 8172 – 8183, 2015.

[110] G. Athanasopoulos, R. Hyndman, H. Song, and D. Wu, "The tourism forecasting competition," *International Journal of Forecasting*, vol. 27, no. 3, pp. 822–844, 2011.

[111] R. Hyndman, "Benchmarks for forecasting." `http://robjhyndman.com/hyndsight/benchmarks/`. Last accessed: 20-08-2016.

[112] M. Gilliland, *Forecasting FAQs*, pp. 193–246. John Wiley Sons, Inc., 2012.

[113] M. Gilliland, *The Business Forecasting Deal: Exposing Myths, Eliminating Bad Practices, Providing Practical Solutions*. John Wiley Sons, Inc., 2010.

[114] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[115] M. Leonard and B. Elsheimer, "Count series forecasting," 2015.

[116] M. E. Silva, "Modelling time series of counts: An inar approach," *Textos de Matemática, Departamento de Matemática da Universidade de Coimbra*, vol. 47, pp. 107–122, 2015.

[117] N. Kourentzes, "Intermittent demand forecasts with neural networks," *International Journal of Production Economics*, vol. 143, pp. 198–206, May 2013.

[118] M. Leonard and B. Elsheimer, "Count series forecasting," tech. rep., SAS Institute Inc., 2015.

[119] ONOS, "Raising the bar on sdn control plane performance, scalability and high performance," tech. rep., ONOS ON.Lab, April 2015.