MULTI-OBJECTIVE FLOW MEASUREMENT IN SOFTWARE-DEFINED NETWORKS (SDN) FOR DATACENTER

HAMID TAHAEI

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2018

MULTI-OBJECTIVE FLOW MEASUREMENT IN SOFTWARE-DEFINED NETWORKS (SDN) FOR DATACENTER

HAMID TAHAEI

THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2018

UNIVERSITY OF MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Hamid Tahaei

Matric No: WHA130058

Name of Degree: Doctor of Philosophy

Title of Thesis: Multi-objective Flow Measurement in Software-Defined Networks (SDN) for Datacenter

Field of Study: Network & Security (Computer Science)

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every right in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:

Designation:

MULTI-OBJECTIVE FLOW MEASUREMENT IN SOFTWARE-DEFINED NETWORKS (SDN) FOR DATACENTER

ABSTRACT

Network traffic is growing exponentially due to the ever-increasing number of users, datacentres, Internet of Things (IoT) devices, and cloud-like applications/services. Network traffic monitoring and measurement has become a vital task and a crucial requirement for Datacentre Networks (DCNs) due to providing fine-grained and timely-based traffic flow information for network applications and management. Traditional network monitoring and measurement techniques either impose extra overhead into the network, or are inaccurate. In reducing the limitations in the traditional flow management systems, the most recent measurement methods elevate the accuracy and alleviate cost issues by applying an emerging technology known as Software-Defined Networking (SDN). SDN has emerged as an evolutionary paradigm in Datacentre Networks (DCN). It enables flexibility by separating the data from the control plane and centralising network decision making, and offers innovation in the network through network programmability. Despite the multitude of efforts proposed for traffic measurement in SDN, current solutions still incur high cost and limitations. These costs are seen as a multi-objective problem as it involves different overheads in the data and control plane such as controller overhead, communication overhead, and message interaction overhead. The problem is even more complex in different network deployments, "in-band and out-of-band". Furthermore, the distinguishing property of SDN is the centralised controller architecture, which results in significant managerial benefits. Due to several scalability and availability issues of a centralised model, such as a single point of failure and network bottleneck, the controller has been made into a decentralised model that is physically distributed. However, little effort has been devoted to measurement techniques in SDN distributed controller architecture. Moreover, the imposed costs of flow measurement in

distributed controller architecture are still an issue that remains unsolved. To address the aforementioned problems, a multi-objective and cost-effective network traffic flow measurement framework was proposed for DCNs. The proposed framework implements SDN capabilities to provide a fine-grained and accurate flow measurement that effectively minimises multi-objective costs for centralised and decentralised SDN controllers in different network deployments. The proposed framework is rigorously evaluated through several experiments, including emulation and simulation. The verification of both experiments is made with current state-of-the-art algorithms. To validate the simulation results, an available dataset from a public datacentre was used. The simulation results were then verified using statistical modelling and t-tests. The results obtained from the various experiments show the effectiveness of the proposed framework and algorithm.

Keywords: traffic Measurement, software defined network measurement, network monitoring, datacenter traffic measurement and monitoring.

PENGUKURAN ALIRAN PELBAGAI OBJEKTIF DALAM RANGKAIAN PERISIAN YANG DITETAPKAN (SDN) BAGI PUSAT DATA

ABSTRAK

Jumlah trafik rangkaian meningkat dengan pesat disebabkan oleh peningkatan bilangan pengguna, pusat-pusat data, peranti-peranti berhubung internet *Internet of Things* (IOT), dan aplikasi perkhidmatan komputeran awan. Pemantauan dan pengukuran trafik rangkaian menjadi keperluan penting kepada Rangkaian Pusat Data *Datacentre Networks* (DCNs) dalam menyediakan maklumat aliran trafik yang baik dan tepat pada masanya untuk pengurusan dan aplikasi rangkaian. Teknik pemantauan dan pengukuran rangkaian sedia ada adalah kurang tepat atau menambah overhed ke dalam rangkaian. Bagi mengurangkan batasan aliran sistem pengurusan sedia ada, kaedah pengukuran terkini dapat menambah ketepatan dan mengurangkan masalah kos dengan menggunakan teknologi baru yang dikenali sebagai Rangkaian Perisian yang Ditetapkan *Software-Defined Networking* (SDN). SDN telah muncul sebagai evolusi paradigma kepada Rangkaian Pusat Data (DCN). Ia memberi fleksibiliti dengan memisahkan data dari aras kawalan dan memusatkan keputusan rangkaian serta menawarkan inovasi dalam rangkaian melalui pemprograman rangkaian. Walaupun banyak usaha yang mencadangkan penggunaan SDN terhadap pengukuran trafik, penyelesaian semasa masih melaporkan adanya batasan terhadap kos. Masalah kos ini dilihat sebagai masalah pelbagai objektif kerana ia melibatkan overhed yang berbeza dalam data dan aras kawalan seperti pengawal overhed, komunikasi overhed, dan interaksi mesej overhed. Masalah ini menjadi lebih rumit apabila pelaksanaan pada rangkaian yang berbeza seperti “in-band” dan “out-of-band”. Tambahan pula, ciri-ciri yang membezakan SDN terletak pada seni bina pengawalan berpusat yang memberi kelebihan kepada unit pengurusan. Oleh kerana beberapa masalah skalabiliti dan ketersediaan pada model berpusat seperti satu titik kegagalan dan kesesakan rangkaian, unit pengawalan telah direka menjadi model tidak

berpusat yang teragih secara fizikal. Walau bagaimanapun, hanya sedikit usaha yang memberi penumpuan terhadap teknik pengukuran dengan menggunakan seni bina pengawalan teragih SDN. Selain itu, kos aliran pengukuran yang dikenakan dalam seni bina pengawalan teragih masih merupakan masalah yang belum dapat diselesaikan. Untuk menangani masalah tersebut, rangka kerja pengukuran aliran trafik rangkaian berbilang objektif dan kos efektif dicadangkan untuk DCN. Rangka kerja yang dicadangkan ini menggunakan keupayaan SDN untuk menyediakan pengukuran aliran yang tepat dan berkesan mengurangkan kos berbilang objektif bagi pengawalan SDN berpusat dan tidak berpusat di dalam rangkaian yang berlainan. Rangka kerja yang dicadangkan dinilai dengan teliti melalui beberapa eksperimen, termasuk emulasi dan simulasi. Pengesahan kedua-dua eksperimen dibuat dengan algoritma terkini. Untuk mengesahkan keputusan simulasi, set data yang tersedia dari pusat data awam digunakan. Keputusan simulasi kemudiannya disahkan menggunakan model statistik dan ujian-t. Hasil yang diperoleh dari beberapa eksperimen menunjukkan keberkesanan kerangka kerja dan algoritma yang dicadangkan.

Kata kunci: pengukuran aliran, pengukuran rangkaian perisian yang ditetapkan, pemantauan rangkaian, pengukuran dan pemantauan pusat data trafik.

ACKNOWLEDGEMENTS

It would have been impossible to finalise this thesis without the help of a number of individuals. I would like to extend my appreciation to those who generously contributed to this thesis.

Special gratitude goes to my supervisor, Associate Professor Dr Rosli Bin Salleh for his invaluable support and excellent guidance throughout my Ph.D. Indeed, his constant faith in my lab work made me eager to go ahead. Profound and special thanks go to my co-supervisor Associate Professor Dr Nor Badrul Anuar Bin Juma'at, who provided me with valuable assistance, technical support, and academic writing at all levels of the Ph.D. journey. To me, he is more a mentor and a friend than a supervisor. I would also like to extend my thanks to Dr Theophilus Benson from Duke University, North Carolina for his constructive comments. I am also hugely appreciative to those who were directly or indirectly involved in this research, particularly Babak Daghighi, Mohamad Habib Ur Rahman, Salman Iqbal, Ibrahim Targio, Ahmad Firdaus Zainal Abidin, Mazrullhisham Yusuf Mohd Zain, Rita Afriani Mohd Yusu, and Azrul Ahmad – your continuous support and kindness shall not be forgotten; please accept my utmost appreciations to all of you. My deepest and heartfelt thanks go to my parents and sister for their unconditional love, devotion, and unbelievable support. They are the most important people in my world. No words can express my feelings. Their sacrifices, care and encouragement made possible for me to complete my journey.

Above all, I would like to thank God for his grace and blessing in allowing me to complete this study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

CAM	:	Content-Addressable Memory
CDS	:	Congestion Detection System
CeMOC	:	Cost-Effective Multi-objective Controller
CPI	:	Cycles Per Instruction
CPU	:	Central Processing Unit
DCN	:	Datacenter Network
FEST	:	Flow Entry Statistics Trigger
HHH	:	Hierarchical Heavy Hitter
ID	:	Intrusion Decoder
IDC	:	International Data Corporation
IETF	:	Internet Engineering Task Force
IF	:	Instruction Fetch
IoT	:	Internet of Things
IP	:	Internet Protocol
ISP	:	Internet Service Provider
LB	:	Load Balancing
MIB	:	Management Information Base
MIPS	:	Million Instruction Per Second
NBI	:	Northbound Interfaces
NMS	:	Network Management System
OF	:	OpenFlow
PA	:	Polling All
PSAMP	:	Packet Sampling
QoE	:	Quality of experience

QoS	:	Quality of Service
RMON	:	Remote Network Monitoring
SDN		Software Defined Networking
SNMP	:	Simple Network Management Protocol
SSR	:	Single Stat-Request
TCAM	:	Ternary Content Addressable Memory
TCP	:	Transmission Control Protocol
TE	:	Traffic Engineering
TM	:	Traffic Matrix
TPS	:	Transaction Per Second
UDP	:	User Datagram Protocol
WB	:	Write-Back

LIST OF APPENDICES

CHAPTER 1: INTRODUCTION

With the rapid growth of datacentres and the continuous thrive of cloud-like services and the Internet of Things (IoT), a traffic measurement system is seen as a necessary requirement for Datacentre Networks (DCN).

Network traffic measurement is a demanding task, and an essential part of a Network Management System (NMS). Network administrators are constantly striving to maintain smooth operation of their networks. If a network were to be down, even for a short period of time, productivity within a company would decline, and in the case of public service departments, the ability to provide essential services would be compromised. Therefore, in order to be proactive rather than reactive, administrators need to monitor traffic movement and performance throughout the network and verify the correctness of states within the network. In other words, the purpose of network traffic measurement is to observe and qualify what is happening in the network traffic with different sizes of magnifying glasses (Mohan et al., 2011).

Likewise, a DCN highly requires accurate measurements of traffic flows in order to effectively monitor the traffic volume in real-time manner. Similarly, a per-flow traffic measurement system can be used to monitor micro-details of every flow in different network layers. Such a system is also known as a fine-grained monitoring system. The fine-grained traffic measurement system in turn needs to include necessary tasks such as Traffic Matrix (TM) estimation, elephant flow detection, and link utilisation to have insight into the network traffic. These measurement tasks are utilized in a wide range of applications, such as network planning, anomaly detection, billing, load-balancing, traffic engineering and security (Chang et al., 2015).

The chapter is organised as follows. Section 1.1. presents the background of the study. Section 1.2 explains the key motivation to carry out the study. In section 1.3, the research gap and the statement of the problem are presented. Subsequently, the objectives of the study and the scope are presented in section 1.4 and 1.5, respectively. Section 1.6 elaborates the methodology of the proposed research. The chapter concludes with providing the thesis layout in section 1.7.

1.1 Background

Traditional flow measurement systems, such as NetFlow (Claise, 2004) and sFlow (Phaal & Lavine, 2004), apply packet sampling approaches to collect information about packets in the network and analyse this information to infer flow-level statistical measurement. They have either a low accuracy or a high deployment cost; moreover they are energy-intensive as they consume more resources (M. Yu et al., 2013). An example of former problems is inaccurate measurement as the result of sampling, because of small flows being missed or multiple monitoring nodes beside the SDN flow path sampling similar packets (Jarschel et al., 2013). An example of the latter is the deployment of NetFlow or a similar sampling-based approach, which requires setting up collectors and analysers. Moreover, enabling NetFlow in the routers may degrade the packet forwarding performance (Cantieni et al., 2006). Furthermore, NetFlow and similar tools such as Sflow, Jflow, IPFIX, and PRTG are hardware-based features that need to be configured to be set for each individual interface on the physical device (switch/router). However, recent measurement methods alleviate the issues of traditional measurement systems such as accuracy and cost, through applying the emerging technology known as Software-Defined Networking (SDN).

The revolutionary SDN architecture has transformed the traditional network design to a potentially flexible and well-managed next-generation of networks that address

problems such as traffic management, analysis, measurement and many others. SDN architecture decouples network control and forwarding functions, enabling network control to become directly programmable. It also abstracts the underlying infrastructure, such as switches and routers, from applications and network services. Such abstraction provides full visibility of network entities, including devices and traffic. In SDN, a central controller collects flow statistics by either directly requesting or passively receiving them from switches. In the direct request approach, which is known as *pull-based*, the controller makes a request by sending a request packet to the switch, and then receives flow information from the switch. In the passive approach known as *push-based*, the controller receives flow information upon expiration of the corresponding flows' entry time-out. The statistics reach the central controller and are used by on-demand applications in the network such as routing, load balancing and many others. This eliminates the sophisticated process of sampling approach for flow level measurement used in traditional methods. However, current methods that apply the push-based approach are inefficient for timely-based flow measurement systems and incur inaccurate flow measurement, as the controller only receives statistical information when a flow entry timeout is reached (Chowdhury et al., 2014). Moreover, implementing the pull-based approach imposes massive costs to the controller's channel bandwidth and processing delay for a single SDN controller, as the SDN controller frequently sends requests and receives replies (H. Xu et al., 2017).

According to the data from Stanford Computer Science and Electrical Engineering; the study in (Naous et al., 2008), with 10 different DCNs, the number of active flows is up to 10,000 with 5,500 active hosts and the average number of active flows at a switch in any time of the second is at most 10,000 flows, respectively. Due to the large-scale DCN traffic and infrastructure, current solutions that implement both of the aforementioned approaches above are still insufficient to deliver on-demand requirements

of DCNs to satisfy a low-cost and timely-basis flow measurement system. The flows in the examined DCNs are generally less than 10Kb in size, and the majority lasts less than a few hundred milliseconds. In addition, new flows can arrive in a fast sequence (10μs) of each other, resulting in a rapid arrival rates. Hence, with regard to this massive scale of DCNs, there are still rooms in SDN that necessitate designing a scalable, cost-effective and accurate timely-basis flow measurement for DCNs.

1.2 Motivation

According to a report by the SDxCentral (SDxCentral, 2016), the SDN market is expected to grow from $1.5 billion in 2013 to $35.6bilion in 2018. Likewise, the International Data Corporation (IDC) (IDC, 2016) recently forecasted that the control layer/virtualization software market as a single segment of the overall SDN market is expected to reach $2.4 billion in 2020. Moreover, the IDC expects that the control layer/virtualization software and SDN applications will observe the fastest growth world-wide, which will be worth approximately $5.9 billion in 2020. Furthermore, SDN is the most rapidly involving landscape, and DCN (cloud computing) is the primary driver of the vast rise in SDN, which expect a market worth more than $12.5 billion in 2020. However, the market and industry observers are still struggling with understanding the potential strength of SDN in traffic measurement, and are apprehensive about the sophistication in a large-scale network.

Unlike traditional networks, the network intelligence is logically centralised in an SDN controller that represents the core of the SDN architecture. Traffic measurement in SDN is entirely dependent on the central controller, and that must always be well-managed for two main reasons: (i) the centralised controller would always remain a hotspot (bottleneck) if the traffic measurement system imposes extra overhead. Therefore, the functionality of the centralised controller may overwhelm; (ii) the accuracy of the SDN

traffic measurement system would decrease significantly if the system was obliged to reduce the overhead.

Despite the promising architecture of SDN and the simplicity offered by this technology, a traffic measurement system was not considered as part of the initial design. Currently there is no built-in traffic measurement system for a large-scale DCN. Recent proposed approaches either present a general mechanism, which is insufficient for a massive size of network with different configuration, or have limitations in terms of identifying various costs imposed by their functionality.

1.3 Statement of the Problem

Next generation DCNs are characterised by their huge scale and the diversity of the generated traffic. One of the crucial tasks and a fundamental requirement for managing these large networks is an accurate per-flow-basis traffic measurement mechanism to monitor traffic volume.

Traditional flow measurement methods in DCN have shown to be costly and inaccurate (Su et al., 2015). Even in SDN, current solutions reported limitations based on the different approaches they implement. For example, a pull-based approach is accurate but imposes extra overheads (costs) in the network (Su et al., 2015). Several efforts have been devoted to overcome different overheads imposed by the pull-based measurement approach, such as those relating to the data and control plane. An example for data plane overheads is communication, which is the amount of network traffic volume incurred by the flow measurement, whereas the number of message interactions and the controller overhead are considered control plane overheads. In contrast, the push-based approach is light-weight in terms of overhead, however, it is incapable of guaranteeing the accuracy (H. Xu et al., 2017). Therefore, the advantage of one approach is achieved at the expense of the other. The situation becomes worse in in-band network deployment when

monitoring and routing traffic shares bandwidth along the same link. This results in a delay of flow statistics to be reached at the central controller because normal network traffic may disturb the flow statistics traffic.

In addition, the distinguishing property of SDN is the centralised controller architecture, which results in significant managerial benefits. However, this property represents a single point of failure. Moreover, like any other centralised system, a fully physically SDN centralised controller is inadequate and introduces issues of scalability, reliability and a performance bottleneck (Dixit et al., 2013). To overcome these obstacles, industry and academia proposed decentralised (multiple) SDN controller designs by which the central controller can be physically distributed but logically centralised (Xie et al., 2015). However, applying a decentralised controller may result in several unexpected performance degradations, such as accuracy, and various overheads in the network and SDN controller. Furthermore, only one controller in the master mode is able to control the switch(es) every time (Pfaff et al., 2012), therefore, selecting a controller for polling switches has an extreme effect on measurement tasks in terms of the costs and accuracy of statistical measurements. In addition, different deployment (i.e. out-of-band and in-band network deployment) of such a scenario has major effects on several factors in the network such as node-to-controller latencies, network availability and performance metrics (Karakus & Durresi, 2017). Therefore, selecting a master controller among multiple controllers fetching flow statistical information plays a vital role in the accuracy of real-time monitoring as well as costs. Furthermore, the synchronisation of multiple controllers in the network causes extra overhead and delay in transferring flow statistics, which may lead the measurement system to being inaccurate or costly.

Therefore, in order to address the absence of a fine-grained traffic measurement system in a decentralised SDN controller scenario, and to overcome the primary challenge of the

flow measurement system (i.e. minimising different overheads while maintaining accurate and near real-time flow measurement as a single problem which can be seen as a multi-objective problem), it is imperative to design and develop a fine-grain cost-effective multi-objective measurement system that supports near real-time flow measurement with high accuracy.

1.4 Statement of the Objectives

The aim of this study is to propose a multi-objective framework for a near real-time fine-grained flow measurement system that can be implemented in a fully centralised or distributed SDN controller design. In order to achieve this aim, the following objectives need to be taken into the consideration.

(a) To study the traditional network traffic measurement and monitoring approaches and perform a gap analysis review on the state-of-the-art SDN techniques for network traffic measurement and monitoring.

(b) To propose a comprehensive mathematical formulation and analysis on different costs such as communication overhead, message interaction, and controller overhead as a multi-objective problem in the context of network traffic flow measurement.

(c) To propose a multi-objective flow measurement framework that effectively minimises the costs and provides near real-time flow statistics in a fully centralised and distributed SDN controller.

(d) To evaluate the performance of the proposed multi-objective framework against similar existing state-of-the-art approaches in SDN.

1.5 Scope of the Research

Flow measurement systems are widely implemented in DCNs. For example, they are used as input for other network applications such as Congestion Detection System (CDS), Load Balancing (LB), Traffic Engineering (TE) and many others. Flow measurement

systems can also be used in different types of networks as well as ISPs, Enterprises, Private Clouds and others. Existing flow measurement solutions mainly deal with the costs and accuracy associated with the measurement of flow statistics. The costs are defined as various overheads imposed by measuring flow statistics in the network and controller. A research contribution highly depends on the defined aim and the predefined target for implementation. This study focuses on traffic flow measurement in DCNs with centralised and decentralised (multiple) controllers to provide inputs for the aforementioned demands. The following presents the scope and limitation of this study:

- This study focuses on SDN flow measurement in DCN with the aim of minimising multi-objective costs on a timely basis and with an approach that is close to real-time.
- The problem formulation of this study is carried out in out-of-band and in-band network deployment. In out-of-band configuration, signalling requires a dedicated network between the controllers and switches, whereas in in-band deployment, transmission of the control and data message takes place in a shared network bandwidth.
- The proposed framework is evaluated in different fat-tree topologies as fat-tree is the most common and a standard topology for datacentre networks.
- The implemented SDN protocol in this study is OpenFlow version 1.3. Currently, OpenFlow version 1.3 is the most prevalent and the *de facto* standard for the commodity switches. However, the implementation and proposed system can be implemented on further version of OpenFlow.
- The performance metric in this study is costs, which is defined as various overheads caused by generating extra traffic and a calculation process to measure flow statistics. Therefore, the metrics are different overheads such as 1) network overhead, which is well-known as communication overhead, that is the traffic imposed by statistical measurement; 2) message interaction overhead, which is the number of messages

required to traverse the network for measurement purpose; and 3) controller overhead, which is the controller workload (CPU) imposed by a calculation of statistics.

- The evaluation of the multiple-controller design takes place in the EC2 Amazon cloud as the experiment required a large CPU and memory power.
- Due to the large scale of the evaluation and limited machine power, the evaluation of the proposed framework with the real dataset is carried out through a trace-driven simulator. However, sufficient statistical modelling is carried out to prove the correctness of the outcomes.
- As the experiment is through emulation, different network latencies are artificially generated for the accuracy-related evaluation.

1.6 Methodology

In setting out to achieve the stated goal of this study, the research methodology is carried out in four phases, as shown in Figure 1.1.

First phase: Identifying the research gap. This stage is explored in Chapter 2, where it starts by presenting information about traditional network traffic monitoring and measurement approaches. It is then followed by presenting a comprehensive background of SDN and an introduction of the original measurement approach proposed by OpenFlow. The phase ends by investigating and categorising the current trends and methods of traffic measurement systems in software-defined networking. The following steps are involved in this phase: (i) investigating different categories of traffic measurement, (ii) reviewing the current state-of-the-art SDN traffic measurement approach, (iii) analysing the current solutions and observing their weaknesses and strengths (gap analysis) (thereafter, the problem statements and research objectives are defined), and (iv) collecting an appropriate network traffic dataset from a valid source.

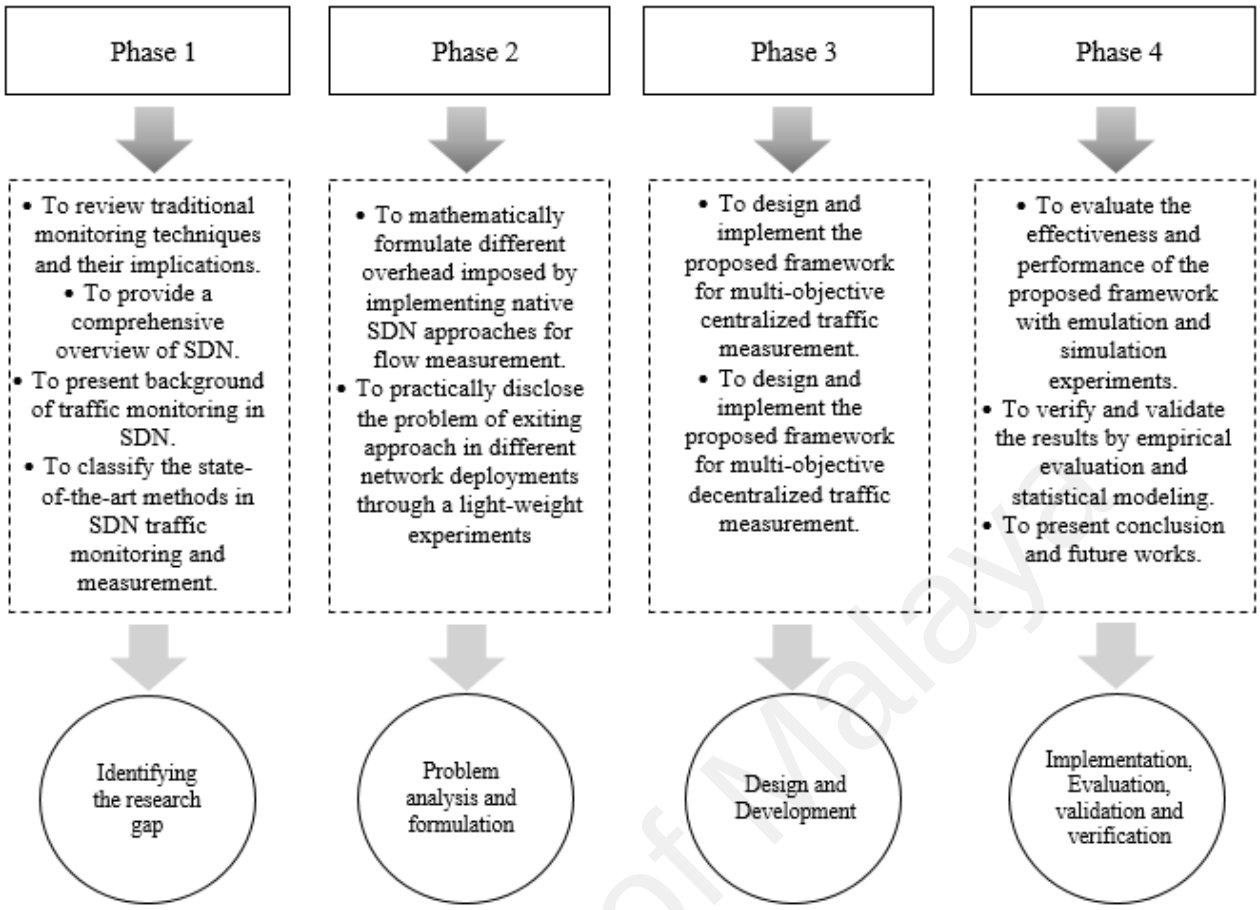


Figure 1.1: Proposed Research Methodology

Second phase: Problem analysis and formulation. This phase is accomplished in Chapter 3, where the problem is analysed, formulated and shown in mathematical notation. Moreover, a mathematical analysis is performed to clarify the problem in different network configurations. Finally, the phase presents a light-weight experiment to reveal the problem experimentally and disclose it through Wireshark captured packets.

Third phase: Design and development. Chapter 4 elaborates on this phase by sketching the initial solution and developing the final design. Firstly, the phase proposes a multi-objective design for network flow measurement for the centralised SDN controller. Secondly, a new design is presented to decentralised (multiple) SDN controller. Finally, the two proposed designs are formed as a unified framework.

Fourth phase: Chapter 5 presents the implementation, evaluation, conclusion and future work. This phase starts by implementing the proposed framework in the lab environment; subsequently the large-scale implementation takes place on EC2 Amazon cloud. This is followed by extensive experiments to evaluate the performance of the proposed framework. The performance evaluation is based on several metrics against the state-of-the-art solutions in SDN measurement. The findings from the simulation are verified using extensive statistical tests. Finally, the conclusion is presented and future works are highlighted.

1.7 Layout of the Thesis

This thesis comprises six chapters. Every chapter of the thesis is divided into three sections; (i) introduction that indicates the objective of the chapter; (ii) body in which the corresponding materials of the objective are described; and (iii) conclusion to summarise and assess the objective to be achieved of the corresponding chapter with a linkage to the next chapter. The remainder of the thesis is organised as follows.

Chapter 2 aims to review existing research in the field of traffic measurement and monitoring systems. It begins with an overview of traditional models of traffic measurement approaches, presenting their pros and cons. The chapter is followed by a comprehensive overview of the SDN architecture, describing different layers and their responsibilities. A brief background of the native approaches for traffic measurement proposed by different OpenFlow specification versions is also presented. The chapter ends with a comprehensive review and classification of existing efforts devoted to SDN traffic measurement and monitoring.

Chapter 3 presents an analysis of the problem to show the impact of different approaches on the traffic measurement outcome. The problem is analysed in

mathematical notation, and a light-weight mathematical analysis is performed to clarify the problem.

Chapter 4 elaborates on the design of the proposed framework in different network model deployments and SDN controller models. Besides, syntax algorithms and flowchart diagram are illustrated to show a detailed process and the interaction between the client and the proposed flow measurement framework.

Chapter 5 presents the implementation and evaluation of the proposed framework. It first explains the experimental setup and the components involved in the extensive experiments. It then explains the findings and compares the proposed framework to similar state-of-the-art methods by means of a comprehensive analysis. Furthermore, a statistical modelling test to verify the findings is presented. The chapter concludes with a comprehensive discussion of the findings.

Chapter 6 discusses the outcomes of the study and how the objectives have been achieved. Subsequently the limitations and delimitations of the proposed mechanism are discussed. The chapter ends with suggestions for future research.

In addition, this thesis has several appendices that includes supportive tables and figures pertaining to the formulations, experiment topologies, and finding verifications.

CHAPTER 2: LITERATURE REVIEW

Network traffic measurement is a key stone of network management tasks (Yuan et al., 2011). Various network management tasks benefit directly from a traffic measurement system. These tasks vary from Traffic Engineering (TE), Load Balancing (LB) and routing decision-making to security and anomaly detections. As such, understanding low-level network transitions is critical for network operators and managers to identify how well their networks are running and consequently what types of services can be offered to the customers based on their network capacities. Therefore, observing and quantifying what is happening in the network is the main purpose of a network measurement and monitoring system, and can be referred to as network visibility. The visibility of a network can be monitored with different size of magnifying glasses which is referred to as granularity, by which all the microdetails of the traffic inside a network can be observed. The granularity varies in accordance with the applied approach. For example, monitoring flow-based network traffic is basically a coarse-grained measurement, which can be more fine-grained by specifying a type of traffic flow.

This chapter aims to conduct a thorough discussion on the major representative research in the area of network traffic measurement. It also provides a comprehensive review on flow-based network traffic measurement approaches in SDN. The chapter starts by giving a broad overview of network traffic monitoring/measurement implications, and by introducing traditional measurement and monitoring methods for network traffic in section 2.1. It then presents an overview of Software-Defined Network (SDN), and introduces different layers and the architecture alongside the underlying fundamental concept, to help readers gain an easy and smooth understanding of SDN. Section 2.3 continues with a light-weight overview of the original SDN measurement approaches introduced by OpenFlow specification 1.3 and 1.5. In section 2.4, the chapter discusses

the state-of-the-art SDN measurement solution and the latest trends in flow-based network traffic measurement in SDN in detail.

2.1 Traditional Measurement and Monitoring Architecture

The goal of measuring and monitoring traffic is to observe and quantify the interaction and transaction in a network. In other words, it discloses what is happening in the underlying traffic of the network by actively or passively gathering data related to the traffic. This information offers supreme opportunities for both end-users and providers. Table 2.1 describes the information offered to different parties through network measurements (Mohan et al., 2011).

Table 2.1: Offered information by network measurement for different parties

	Goal	Measure
Provider (e.g., DCN, ISP and etc)	• Capacity planning • Operations • Value-added service (e.g., customer reports) • Usage-based billing • performance tuning • Planning	• Bandwidth utilisation • Packet per second • Round trip time (RTT) • Packet loss • Reachability • Routing diagnosis
End-users	• Monitor performance • Plan upgrades • Negotiate service contracts • Optimise content delivery • Usage policing	• Bandwidth availability • Response time • Packet loss • Connection rate • Service quality • Host performance

Table 2.1 presents various tasks of a measurement system for two parties such as provider and end-user, and describes the goal and measurement criteria for each party. For example, transferring the maximum amount of data in the minimum time might be

interesting for providers. Usage billing is one of the most important aspects of a provider's career that emerge out from network traffic measurement. In addition, datacentre /ISP providers might benefit from upgrading their plans and offers to customer (end-users). From the end-users' point of view, a persistent connection with full bandwidth might be a crucial requirement.

Network measurement is broadly categorised into two main categories, namely passive and active measurement. Below these two categories and their usage, along with the advantages and disadvantages are described.

2.1.1 **Passive Measurement**

Passive network measurement records the existing network traffic and analyses data by using extra hardware and devices such as link splitters/hubs. This approach passively listens to the network traffic in two ways, either by a) duplicating the traffic on each link/interface (i.e. switch or router interface) and sending it to a collector or analyser, or b) reading "switches/routers" buffer. Figure 2.1 shows an example of a passive network measurement schema. Passive network measurements are commonly collected in four ways: (1) polling management information base (MIB) data from routers, (2) packet monitoring, (3) flow monitoring and (4) sampling.

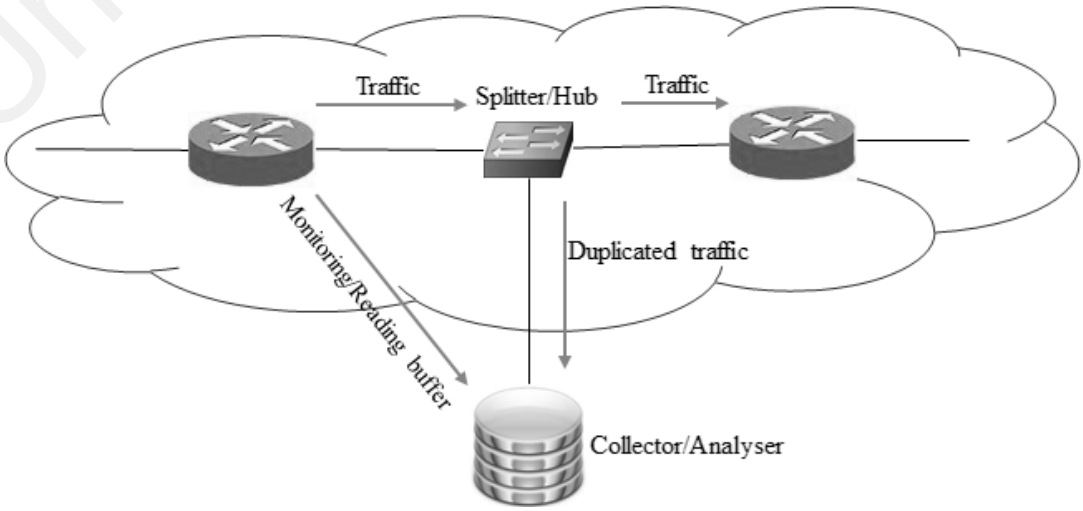


Figure 2.1: Example of passive network measurement schema

2.1.1.1 **MIBs and SNMP Statistics**

MIB is a database in which traffic statistics of the network are retained. The statistics in MIB are coarse-grained and can be queried by routers. MIB-11 is a standardised version of MIB that is available in most of the network elements. MIB-II offers traffic statistics such as transmitted packets, byte counters at interface and counters of packets, and bytes lost. However, these statistics are highly aggregated and cannot be considered as fine-grained network statistics. SNMP (Case et al., 1990), a simple network management protocol, is used for polling the routers for recovering (querying) these information. However, to prevent performance degradation of network devices and any impairs, the SNMP statistics are commonly polled every five minutes, although polling SNMP from routers even in intervals of a few seconds is claimed to not impair routers' performance (Case et al., 1990). Remote Network Monitoring (RMON) (Waldbusser, 2006), is another standardised protocol of MIB that was designed for remote monitoring. Network devices such as routers can record and query traffic statistics and network conditions using RMON by configuring a remote agent inside the devices. The remote agent provides network traffic information and conditions such as captured packets, events, filters, and raises alarms based on some predefined thresholds. However, the implementations of RMON are shown to be limited to low speed interfaces, as its adaptability and the various range of task makes it complex and unsuitable to continually measure and export detailed traffic data.

2.1.1.2 **Packet Monitoring**

Packet monitoring is accomplished by duplicating a stream of packets from the interface(s) of the network devices. Thereafter, different processes such as selecting, storing, analysing and/or exporting various information are performed on these duplicated packets. There are three approaches to packet monitoring: (1) monitoring the duplicated physical signal on a separated interface. Some hardware such as optical splitters can copy

signals on a medium and bring them to another interface to monitor the signals that carry packets; (2) monitoring packets of the traffic on a shared medium; (3) attaching devices for monitoring traffic packets that have been duplicated on a separate interface through a router/switch.

However, the main shortcoming of packet monitoring is the resource constraint. Due to the heavy traffic volume and full line rates of high speed links, monitoring packets seem unsuitable with the current resources. A common solution to overcome this issue is to restrict packet monitoring to some initial number of bytes in the packets to control data bandwidth at the monitor (Micheel et al., 2001). This is reasonable solution, since the IP header and other protocol header information is located at or near the start of the packet. Even so, widespread continuous collection, transmission and storage of unreduced packets have been infeasible for a number of years due to the immense volumes of data relative to the capacity of the system to collect them. Collection of full packet header traces is feasible only for limited durations. Instead, for applications that require continuous monitoring over an extended period, it is common to perform analysis at or near the monitor by forming flow records or other aggregate statistics, or more general stream querying functionality (Iannaccone et al., 2004). Collection of packet IP and transport headers is commonly performed using tcp-dump or its variant windump. Depending on the traffic load and processing power at the measurement host, these tools may also be able to capture parts of packet payload.

Several factors limit the deployment of packet monitoring e.g. equipment (devices), availability and administrative cost. A more recent approach to packet monitoring is to embed the passive measurement functionality within network elements such as routers and switches. Once network elements are equipped with packet monitoring capabilities, the measurement of packets can become ubiquitous. However, due to a lack of additional

computational resources for packet measurement, network elements such as routers and switches may face restrictions in performing measurement analysis. To address these restrictions, some form of data reduction is required, both in the selection of information of packets and in the selection of packets to be reported on. As an example, some packet sampling capabilities are becoming available on routers, such as InMon sFlow (Panchen et al., 2001). Packet selection capabilities for network elements was standardised by packet sampling (PSAMP) by the Working Group of the Internet Engineering Task Force (IETF). The main goal of IETF is to standardise a set of packet selection capabilities that are simple enough to be ubiquitously deployed, yet rich enough to support the need of measurement-based network management system application.

2.1.1.3 **Flow Monitoring**

A flow of network traffic is a set of packets with a common property, known as the flow key, that is seen within a period of time. Many routers construct and export the summary of statistics of the packet flows that pass through them. Ideally, a flow record is assumed to be a summarising set of packets that arises in the network through some higher-level transaction, for example, a remote terminal session or a Web-page download. In practice, packets are formed as a flow depends on the algorithm used by the router to assign a packet to a flow. Flow key is specified by fields from the different packet header fields, such as the IP source and destination address and TCP/UDP port numbers. Flows in which the key is specified by individual values of these fields are often called raw flows, as opposed to aggregate flows in which the key is specified by a range of these quantities. Flow statistics are created as follows. A well-known flow monitoring tool is Netflow (Claise, 2004), that was originally developed by Cisco to provide a way to collect statistics about individual IP flows in a data network. In NetFlow, each switch or router, maintains a flow cache that tracks flow statistics for each flow, usually identified by 5-tuple (source and destination IP address, source and destination TCP/UDP port, and IP

protocol number) and type of service. As each packet arrives, its header fields are checked to see if it matches an existing entry in the flow cache. If it does, then the flow cache entry is updated appropriately, i.e., by incrementing the packet and byte counts. If the flow is not already present in the flow cache, a new entry in the flow cache is created. NetFlow has four policies to decide when to send the flow record to a NetFlow collector: (1) when a TCP packet is seen with a FIN or RST flag indicating flow completion, (2) when a flow idle timeout expires, (3) when a hard timeout fires indicating that the flow has been tracked for γ seconds regardless of whether it is still sending traffic, and (iv) when the flow cache is full and an entry must be evicted. When any of these four conditions hold, the switch sends a NetFlow record including flow statistics to a collector for further analysis.

However, Implementing NetFlow in hardware requires a dedicated Content-Addressable Memory (CAM) to track this information at line-rate. This hardware is not found in all switches and support for NetFlow is chiefly found in Cisco products. Further, NetFlow timeouts are specified at second granularity and in practice many implementations do not allow for values less than 30 seconds (Suh et al., 2014).

2.1.1.4 **Sampling**

Sampling is another way of passive measurement that can significantly reduce the amount of data imposed by the measurement method. It can be used when a full analysis of network traffic is not required. In this approach, a few packets are chosen as a sample of a probabilistic traffic. However, since sampling is a probabilistic monitoring approach, an error ratio is expected. As an example of sampling tool, Imon sFlow (Panchen et al., 2001), (Phaal & Lavine, 2004) aims to provide fine-grained network measurements without requiring per-flow state at switches. Instead it relies on two forms of sampling: packet sampling and port counter sampling. For packet sampling, the switch captures one

out of every *N* packet on each input port. It then immediately forwards the sampled packet's header encapsulated with metadata to a central collector. The metadata include the sampling rate, the switch ID, the timestamp at the time of capture, and forwarding information such as input and output port numbers. It is worth mentioning here that *N* can be configured per-port and needs not be the same for all ports.

The rate of samples produced by sFlow is not constant; it is equal to the packet rate on the port divided by the sampling rate. Since the packet rate varies dramatically based on network load and packet size, the rate of samples also varies. Note that a packet passing through multiple switches is eligible to be sampled by every switch along the path. If a flow passes through *k* switches, combining the samples from those switches gives an effective factor of *k* increase in the sampling rate. From the gathered samples, the collector can probabilistically infer a number of flow statistics, i.e., it can estimate the number of packets and bytes in each flow by simply multiplying the number of sampled bytes and packets by the sample rate, N (Phaal et al.). This approach is called simple scaling and is an unbiased estimator for the actual number of bytes and packets sent by the flow. The technique is also referred to as Maximum Likelihood Estimation (MLE) by which it estimates the byte and packet counts of the flow.

This simple scaling approach has the limitation that it requires a large number of samples to provide accurate estimates of the true flow byte and packet counts. The expected relative error is inversely proportional to the square root of the number of samples, *s*, gathered from that flow. In particular, the expected error in percent can be estimated as shown in equation 2-1 (Phaal et al.).

$$Percent\ Error = \leq 196 \times \sqrt{\frac{1}{s}} \qquad \textbf{2-1}$$

An analysis of real datacentre workloads by Benson et al (Benson et al., 2010) found that an average of 3,000 packets and 60 flows arrive at each top-of-rack switch in any given 100ms window. This means the average flow has 50 packets in a 100ms window. Even if all 50 packets from a given flow are sampled, it can only estimate the flow's actual rate with approximately 30% error. In practice with realistic sampling rates, even this is optimistic. Using the maximum likelihood estimation approach, there are only two ways to improve accuracy: (1) increase the sampling rate and/or (2) increase the sampling period (Suh et al., 2014).

However, increasing the sampling rate is difficult. The sample rate peaks at between 300 and 350 samples per second. It is believed that this limit is a consequence of the switch's control CPU being overwhelmed. With a limit of 350 samples per second, the expected number of samples for a given flow in a 100ms time window that samples from 60 flows is less than one. While newer switches may provide faster control CPUs, it seems likely that it will be infeasible to get enough sFlow samples in a short period, i.e., 100 ms, to provide an accurate estimate of the flow throughput for the foreseeable future (Suh et al., 2014).

Also, Netflow introduced a new version called "Sampled NetFlow" (Cisco), mode that produces NetFlow records based on sampling *1* in *N* packets that traverse a switch rather than every packet. However, the samples are still applied to the records in the flow cache and records are still sent according to the same policy. Thus, Sampled NetFlow incurs the same coarse-grained timeouts that make NetFlow unsuitable for low-latency monitoring.

2.1.2 **Active Measurement**

In active measurement, probe packets are continuously sent across network paths through which the end-to-end performance properties of the network can be monitored. In other words, this measurement approach generates additional traffic to monitor and

measure the network properties. Active measurement requires careful planning before deployment in the network, as the bandwidth reserved for the probe packets is limited to less than five percent of the path's total capacity (Mohan et al., 2011). This is the case in most continuous SLA-measurements, meaning the test traffic and customer traffic share the same bandwidth. Therefore, the extra traffic generated by this approach may disturb the normal network traffic and cause congestion/saturation and packet-loss in the network. Active measurement is used for different ranges of network monitoring purposes such as packet-loss, round trip time, one-way-delay, end-to-end connectivity and available bandwidth detection.

However, since probes can be launched from any accessible host, this approach is well-suited for end-to-end performance measurement. End-to-end packet loss can be inferred from gaps in probe sequence numbers observed at the destination, while end-to-end delay is determined by comparing time stamps placed in each probe by source and destination. Packet content is of interest insofar as it influences performance characteristics such as different treatment by routers of packets based on their IP header fields, i.e., the type of service field. Figure 2.2 shows an example of active measurement schema.

Unlike passive measurement, active measurements do not require huge amounts of storage space and they can be used to measure things that are infeasible by using passive measurements. Also, when using active probing, there are no privacy issues since the data used does not contain any private information. All active probe packets are artificial, i.e. they are generated on demand and thus they usually contain only random bits as payload. The example presented in Figure 2.2 shows how active probing can be used to measure the response time of a web server. A measurement device or a software agent installed on a normal PC sends web page requests across a network and records the response time.

The most well-known active measurement tools are probably *traceroute* and *ping* which are built in to most operating systems.

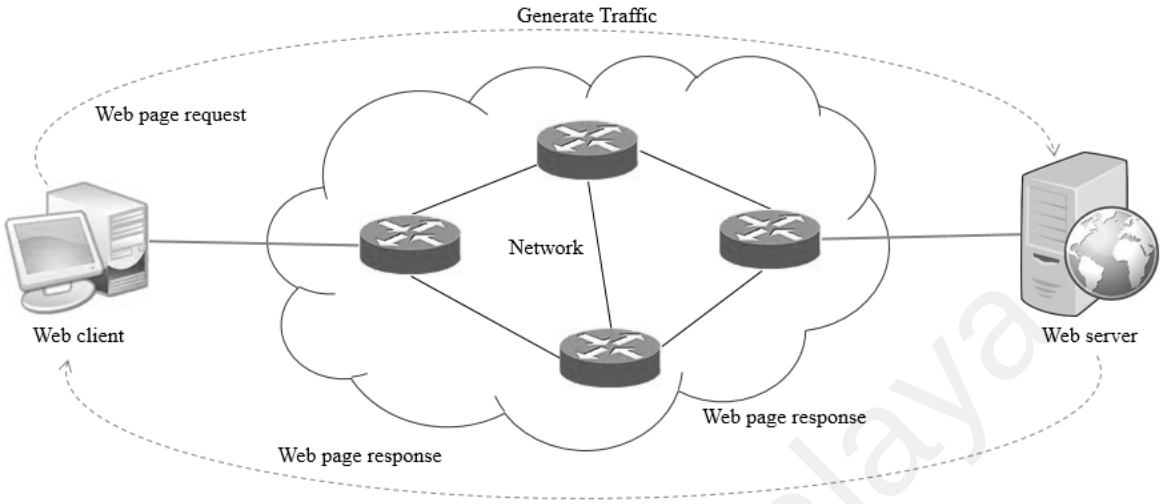


Figure 2.2: Example of active network measurement

These tools such as ping and traceroute allow users to measure roundtrip performance from a host without requiring privileged access to routers in the network interior. Although ping and traceroute require the destination to respond to Internet Control Message Protocol (ICMP) packets, an ability which may be administratively disabled. Also, bulk throughput can be estimated using the *treno* tool (Matt Mathis, 1996), which creates a probe stream that conforms to the dynamics of TCP (Duffield, 2004).

2.1.3 Passive Measurement vs Active Measurement

Active and passive measurements produce different kinds of information and the results do not necessarily correlate well. A more complete picture of the health of a network can be gained by combining the results from both active and passive measurements that is referred to as hybrid measurements. Table 2.2 shows the main difference of active and passive measurement.

Table 2.2: Difference between passive and active measurement

	Active Measurement	Passive Measurement
Capture points	×	✓
Generate additional traffic	✓	×
Accuracy	×	✓
Planning and deigning before deployment	✓	×
Huge storage	×	✓
Same administration domain and permission	×	✓
Extra hardware	×	✓
Extra software/agent	×	✓

As shown in table 2.2, passive measurement is bounded to the points in the network for measurement purpose. Therefore, it is best suited to the situations where the capture points can be freely selected. This is true in situations where the whole network is owned and operated by a single organisation, i.e., corporate premises networks. This allows traffic to be captured from any point along the path from the sender to the receiver. In addition, passive measurements send captured traffic for further analysis to third party devices which needs huge storage capacity. Moreover, extra hardware and agents are required to infer and analyse the information captured by passive measurement. In situations where it is infeasible to select capture points freely, active measurements must be used. This is often the case when measuring delay performance of a VPN that is carried over multiple ISPs.

Active measurements generate additional traffics by sending probes through the networks. It can be made over a network path which are not controlled by the network (Mohan et al., 2011). For example, the *ping* tool can be used from diverse network with different administration domain and permission. On the other hand, passive measurement requires the same administration domain and permission. When it comes to accuracy of the measurements, passive methods are often more accurate. For example, packet loss can

be measured very accurately by monitoring router buffers along the network path. Also, available bandwidth can be accurately measured by monitoring link usage on routers. Also, there are a number of statistical challenges in sampling and analyzing network measurements:

- The majority of available data already have been sampled during collection. For the reasons described in the previous section, raw unsampled data are increasingly difficult to come by, so it is natural to ask, what does the sampled data tells us about the original network traffic?
- Implementations of sample designs may be limited by technology and resources. Technological constraints may limit the ability to use the sample design that is ideal from the purely statistical point of view. Equipment vendors may implement different realisations that approximate the ideal. What are the ramifications for statistical analysis and how do the results of analysis depend on the implementation details?
- Measurements themselves travel from the observation point (i.e., a router in the network) through a number of subsystems to the eventual data repository, possibly with some preprocessing or aggregation on the way. Each stage in the journey presents an opportunity for sampling. At which stage is sampling best performed?
- Best choice of sample design depends on the traffic characteristics. Experimental studies show that network traffic exhibits dependence and rate fluctuations over multiple time scales, leading to heavy-tailed distributions for some traffic statistics. Sample design needs to take account of such behavior, for example, to control estimation variance.
- The best choice of sample design depends on the statistics needed by applications. There is no general agreement on which set of traffic statistics is most useful for network management. Whereas it is possible to optimise the sample design with respect to estimation of a given set of statistics, the design may be suboptimal for

another set of statistics that could play an important role for some future application. For this reason, analyzing the trade-offs between statistical efficiency and flexibility is an important task for sample design.

In reducing the limitation in the traditional flow management systems mentioned above, the most recent measurement methods elevate the accuracy and alleviate the cost (overheads such as generating probes in active approach and traffic duplication in passive approach) issues by applying the emerging technology known as Software Defined Networking (SDN). In the next section, a comprehensive overview of SDN is given followed by the State-of-the-art measurement in SDN.

2.2 Overview of the SDN Architecture

Software-Defined Networking (SDN), is a programming approach that supports decoupling of the control and forwarding plane (Haleplidis et al., 2015). OpenFlow (OF) (McKeown et al., 2008) is the first implementation of SDN; it was initiated in 2008 as a project at Stanford University by Professor Nick McKeown who put forward the concept of SDN (Haleplidis et al., 2015). In the same year, ACM SIGCOMM published a paper titled "OpenFlow: Enabling Innovation in Campus Networks" (McKeown et al., 2008). This paper introduced the concept of OpenFlow in detail. In December 2009, the first version of OpenFlow specification 1.0 was released for use in commercial products. In March 2011, Professor Nick McKeown et al. were again responsible for the inception and establishment of the Open Networking Foundation (ONF), which focused on the development of SDN architecture. In April 2012, ONF released a white paper on SDN titled “Software-Defined Networking: The New Norm for Networks” (Fundation, 2012), where the three-layer SDN architecture was introduced and gained widespread recognition in industry and academia. Although SDN is not restricted to OpenFlow, other control plane decoupling mechanisms existed before OpenFlow. OpenFlow is often considered the standard communication protocol to configure and monitor switches in

SDNs. Figure 2.3 shows the SDN layer architecture. The ONF constitutes of six core organisations, namely Google, Facebook, Verizon, Deutsche Telekom, Microsoft, and Yahoo, and has currently reached more than 100 members with several versions of OF being released under ONF such as 1.1, 1.2, 1.3, 1.4 and 1.5.

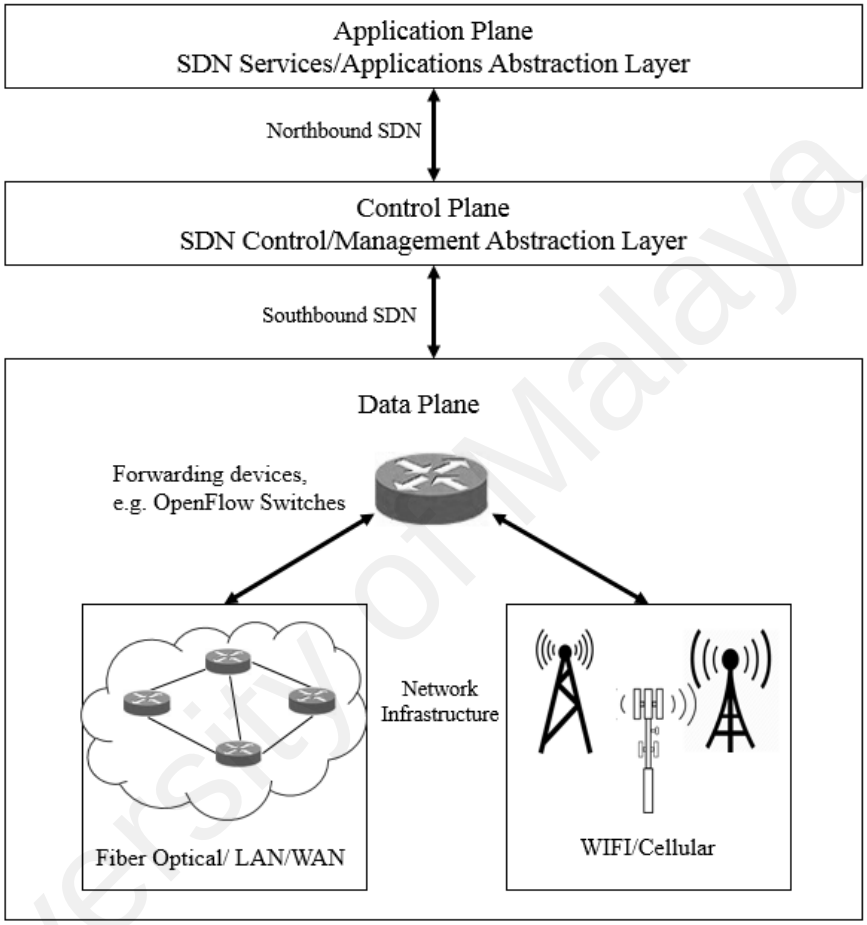


Figure 2.3: SDN layer architecture

The OpenFlow concept is no longer just a research model that can remain within the boundaries of academia, but has been rapidly moved to the production environment. In April 2012, Google announced that its backbone network has been fully operational in OpenFlow, with 10Gbps network links located in 12 data centres around the world. After the implementation of SDN, the utilisation of the WAN lines has increased from 30% to near saturation. Later in April 2013, big companies such as Cisco, IBM, Microsoft, Big Switch, HP and Red Hat worked together to develop SDN applications and established

the OpenDaylight (Medved et al., 2014) controller, which is an industrial-grade open source SDN controller.

2.2.1 SDN Architecture

The SDN architecture (Bozakov & Sander, 2013) consists of three main components: (1) SDN ***application plane*** in which all network applications are executed; (2) SDN controller, which is called the ***control plane***; and (3) SDN devices (switch, routers), which refer to the ***data plane***. The main feature of the SDN architecture is that the controller and data layer are decoupled and abstracted from each other. In addition, programmability is a key feature that enables users to develop their own applications at the application layer using a ***northbound interface*** that provides a programmable API and high-level policy applications and services. Moreover, the ***southbound interface*** provides standard APIs that facilitate the communication between the controller and the switch via ***OpenFlow protocol***. The next section discusses the SDN architecture components in detail.

2.2.2 Application Plane

The application plane is also known as application layer. It consists of various network application services (Feamster et al., 2014) that run on top of the SDN controller. It interacts with the controller through the northbound API interface. These application services can be used to configure the flows to be forwarded based on the changes in the network. For example, load balancing application distributes the traffic across multiple servers or paths according to the current load status. SDN applications communicate with the SDN controller via APIs to manipulate network information. These APIs depend on the controller itself, i.e. on whether the controller provided reaches APIs that enable developers to design their applications. Usually most of the open source and commercial controllers provide REST-FUL-API (Zhou et al., 2014) that can easily be enabled to use any language.

2.2.3 **Northbound Interfaces (NBI)**

NBI (Zhou et al., 2014) is a layer that sits between the SDN controller and high-level services and applications to enable an exchange of information between the controller and network applications. Each controller provides an API interface to allow the user to interact with the lower level details of network functions. For example, controllers such as OpenDaylight, Floodlight, and Ryu define their own APIs that depend on the programming language deployed to extend the controller functionalities, but most of them provide REST-API. Therefore, the Open Networking Foundation (ONF) founded a NBI working group that aims to develop standards for the interface that can be used by all controllers. Recently a number of domain languages such as Frenetic and Pyretic have been introduced to abstract the inner details of the controller and the switch.

2.2.4 **Control Plane**

SDN controller is a network operating system (Clayman et al., 2016) that views a comprehensive network topology and manages OpenFlow switch via a secure communication channel. It is responsible for managing, controlling, and manipulating flow tables (Kuźniar et al., 2015) in the switch. SDN controller communicates with two interfaces, a southbound and a northbound interface. The northbound interface provides programmable API that interacts with the application layer, while the southbound interface communicates with the data layer via a secured channel. A programmable API (Jarschel et al., 2014) provides an abstract view of the network and delivers specific network functions in order to fulfil the network operator's needs. Server messages are interchanged between the controller and data layers via a southbound interface for establishing a connection and retrieving information. For example, SDN Controller manages the forwarding table for each switch based on the header of the packetin message that is sent from the switch. The controller then replies to this message by sending "PacketOut" that informs the switch on how to deal with this packet based on the network

policy. SDN supports two modes of deploying a controller, centralised mode whereas one controller can manage the entire network, and distributed mode where two or more controllers control the whole network. Each controller, called the domain controller, is responsible for managing a number of switches and shares the network information with the other controller. Another mode of distributed controller (Schmid & Suomela, 2013) is the *master/slave* mode where the *slave* controller serves as a backup to the *master* controller in case of any failure. Two metrics are taken into account when measuring a controller's performance: flow setup time and the number of flows per second that the controller can handle. These metrics have a strong influence on the deployment of additional SDN controllers. To date, different types of SDN (compatible) controllers have been developed. Table 2.3 presents some popular SDN controllers with corresponding brief descriptions.

Table 2.3: The SDN Controller and Description

SDN Controller	Open Source	Language	Description
NOX (Gude et al., 2008)	✓	C++, Python	The first proposed introduced controller for OpenFlow to support fast-asynchronous IO.
POX (Mccauley, 2014)	✓	Python	Propose better performance over NOX.
Maestro (Ng et al., 2010)	✓	Java	Presents abstraction view of related state of the network and group into subset.
Floodlight (BigSwitchNetworks, 2016)	✓	Java	Manages OpenFlow and non-OpenFlow protocols.
Beacon (Erickson, 2013)	✓	Java	Presents a cross-platform and modular based controller that supports event-based and thread operation.
OpenDayLight (Medved et al., 2014)	✓	Java	It deploys OSGi framework and provide REST API having weak consistency.

Table 2.3, continued

Trema (Khattak et al., 2014)	✓	Ruby, C	Presents a programming framework that users can develop and test OpenFlow controller on a laptop.
RouteFlow (Nascimento et al., 2011)	✓	C++	Presents virtualized IP routing over OpenFlow hardware.
Ryu (Khondoker et al., 2014)	✓	Python	Supports different version of OpenFlow and integrates with open-stack, building virtual network without VLAN
FlowVisor (Sherwood et al., 2009)	✓	C	Special OpenFlow controller for network virtualization
SNAC (Padhi et al., 2006)	×	C++	Based on NOX-0.4 to manage network, configure devices, and different event monitoring
Helios	×	C	Provides shell programming to perform integrated experiments.
ONOS (Berde et al., 2014)	✓	Java	Building networks for service providers with performance, scale-out design and high availability.

2.2.5 **Southbound Interfaces (SBI)**

SBI (Ros & Ruiz, 2014) enable the SDN controller to manipulate the behaviour of the data plane and make changes according to real-time demands and needs. The main function of SBI is to facilitate communication between a controller and a network switch (both physical and virtual) so that the switch can discover network topology, define network flows and implement requests relayed to it via Standard API. Several standards are available such as DevoFlow, OF-Config and Cisco's OpFlex. Cisco OpFlex is the most popular standardised southbound API for OpenFlow.

2.2.6 **Data Plane**

The data layer consists of a set of networking equipment (such as switches, routers, and middlebox), known as OpenFlow switches, which communicate to formulate a single

network. The OpenFlow switch is responsible for capturing, manipulating, and matching packets against flow table entries. The main function of the SDN switch is to process the transit traffic based on the controller's policy, which decides what to do with packets headed to an ingress interface. It manages a number of flow tables, and each flow entry is associated with a set of instructions or actions that change a packet. When an incoming packet matches the rule in the flow entry, an action is required. The action might be forwarding a packet to a specified port or dropping the packet. OpenFlow involves two types of actions: required and optional (Shahmir Shourmasti, 2013). A required action must be supported in switches, whereas optional action is set based on the network requirements and could be a query by an OpenFlow controller. In addition, the OpenFlow switch supports multiple flow tables and a different group table that sometimes refers to an OpenFlow pipeline (El-Azzab et al., 2011), in which a packet interacts with these flow tables. There are two types of SDN switches, pure (OpenFlow-only) and hybrid (OpenFlow-enabled) (Azodolmolky, 2013). Pure OpenFlow switches have no legacy features or onboard control. These switches completely rely on the controller to forward decisions. Hybrid switches support OpenFlow as well as traditional operation and protocols. There are two approaches to manage flow tables in OpenFlow specification, Proactive Flow (Lin et al., 2013) in which the controller sets up flows in advance, and Reactive Flow (Dusi et al., 2014), where the controller responds to packetin events and dynamically updates the flow table.

2.3 Background in SDN network traffic measurement

In OpenFlow, the monitoring task is accredited by the controller that is connected to all switches via a secure channel interface called the southbound interface. The secure channel is established over a TCP connection between the controller and the switch. The controller accumulates the real-time flow statistics from the corresponding switches, and combines the raw data to deliver interfaces for upper-layer applications. When a switch

receives the first packet of a new flow in the network, it first checks its flow table to find a match for the flow. Then the flow is forwarded according to the corresponding flow entry of the flow table. In the case of a table miss, when there is no match for a flow, the switch forward the first packet header to the OpenFlow controller by a packet_in message. The controller processes the packet header and takes further actions such as setting up the routing path. The controller instructs the corresponding switches along the path to enable a flow by a packet_out message. It is worth mentioning here that OpenFlow specification version 1.0 proposes twelve fields to match. However, the newest specification, version 1.5.1 at the time of writing this thesis, introduces 44 match fields.

According to OpenFlow specification 1.0 (Pfaff et al., 2009), a naive approach to obtain the flow statistics in the network is to query them from the switch through the controller using a single stat-request called Single Flow Request (SSR). This approach is also well-known as pull-based approach and widely used in the literature. This way, fine-grained per-flow information about a predefined individual active flow is requested with the "ofp_flow_stats_request" stats request type. Figure 2.4 shows the structure of a flow statistic request.

```
/* Body for ofp_multipart_request of type OFPMP_FLOW_DESC & OFPMP_FLOW_STATS. */
struct ofp_flow_stats_request {
    uint8_t table_id;         /* ID of table to read (from ofp_table_desc),
                                 OFPTT_ALL for all tables. */
    uint8_t pad[3];           /* Align to 32 bits. */
    uint32_t out_port;        /* Require matching entries to include this
                                 as an output port.  A value of OFPP_ANY
                                 indicates no restriction. */
    uint32_t out_group;       /* Require matching entries to include this
                                 as an output group.  A value of OFPG_ANY
                                 indicates no restriction. */
    uint8_t pad2[4];          /* Align to 64 bits. */
    uint64_t cookie;          /* Require matching entries to contain this
                                 cookie value */
    uint64_t cookie_mask;     /* Mask used to restrict the cookie bits that
                                 must match. A value of 0 indicates
                                 no restriction. */
    struct ofp_match match;   /* Fields to match. Variable size. */
};
OFP_ASSERT(sizeof(struct ofp_flow_stats_request) == 40);
```

Figure 2.4: Structure of Flow Statistic Request (Pfaff et al., 2012)

The predefined active flow is queried based on the exact match of several fields such as input switch port, source/destination address, etc. In OpenFlow specification 1.0, there are twelve flow match fields as shown in Figure 2.5. However, the number of flow match fields in OpenFlow specification 1.3 (Pfaff et al., 2012) is forty. Flow information along with its statistics is sent to the controller by the relative corresponding switch. Figure 2.6 shows the structure of a flow reply message. To query each active flow every time, two messages are transferred in the network; one is a request message from controller to switch and one is a reply message from switch to controller.

Ingress port	Ether src	Ether dst	Ethern type	VLAN id	VLAN priority	IP src	IP dest	IP proto	IP ToS bits	TCP/UDP src port	TCP/UD P dst port

Figure 2.5: OpenFlow Flow Match Table

```
/* Body of reply to OFPMP_FLOW request. */
struct ofp_flow_stats {
    uint16_t length;            /* Length of this entry. */
    uint8_t table_id;           /* ID of table flow came from. */
    uint8_t pad;
    uint32_t duration_sec;      /* Time flow has been alive in seconds. */
    uint32_t duration_nsec;     /* Time flow has been alive in nanoseconds beyond
                                   duration_sec. */
    uint16_t priority;          /* Priority of the entry. */
    uint16_t idle_timeout;      /* Number of seconds idle before expiration. */
    uint16_t hard_timeout;      /* Number of seconds before expiration. */
    uint16_t flags;             /* One of OFPFF_*. */
    uint8_t pad2[4];            /* Align to 64-bits. */
    uint64_t cookie;            /* Opaque controller-issued identifier. */
    uint64_t packet_count;      /* Number of packets in flow. */
    uint64_t byte_count;        /* Number of bytes in flow. */
    struct ofp_match match;     /* Description of fields. Variable size. */
    //struct ofp_instruction instructions[0]; /* Instruction set. */
};
OFP_ASSERT(sizeof(struct ofp_flow_stats) == 56);
```

Figure 2.6: Structure of Flow Reply Message (Pfaff et al., 2012)

Another approach to receive flow statistics is to request all the active flows in the flow table of a switch. This strategy can fall under the pull-based approach, as it queries flows from the controller. In this approach, a request message is sent to the target switch without specifying a particular match. In other words, the request message set the flow match as

a wildcard that as be defined by "*ALL*" in the flow match. Therefore, the reply message contains all the active flows in the switch's flow table. This approach is referred to in the literature as polling all "PA".

The third approach is known as push-based approach, in which the controller receives reports and statistics of each active flow from devices. Each switch sends the statistics of a flow to the controller whenever the flow's time-out is reached, which means the flow entry is expired. This approach reduces the overhead considerably as it eliminates the needs for requesting each flow record, and thus saves on resource utilisation in devices as well as on network overhead. However, the main drawback of the push-based approach is that the controller is not sent flow reports and statistics before time-out entry. Therefore, the approach is neither able to meet the requirements of a system for scheduling purposes, nor those for timely-based monitoring and measurement.

In OpenFlow specification version 1.5 (Consortium, 2014), a new push-based approach was introduced later to retrieve flow statistics by triggering one or several thresholds. This approach was proposed to reduce the imposed overheads by polling flow entry from switch. The mechanism relies on the predefined thresholds, by automatically sending statistics to the controller whenever the thresholds are triggered. However, the approach can neither be implemented as fine-grained nor as timely-based measurement. Also, it can introduce more overheads if the predefined thresholds are too small, or miss timely-based flow measurement for light-weight flows if the threshold is too big.

2.4 State-of-the-art SDN Measurement Solution: A complete overview

Next generation networks are characterised by their huge scale and the diversity of the generated traffic. It is not an easy task to predict the needed traffic measurement characteristics in such networks without sufficient measurement data about individual components in each part of the network. As discussed in previous sections, traditional

traffic measurement implements two approaches, namely active and passive strategies for monitoring/measurement purposes. However, these approaches are insufficient for next generation networks, where the traffic changes dynamically and its volume increases continuously. On this huge scale, the network management application can play a critical role in avoiding unexpected dilemmas. Several SDN solutions have been introduced to address the limitations and sophistications of traditional network traffic measurement approaches mentioned earlier in section 2.2, by efficiently utilising the flexibility of SDN to offer programmable interfaces to attain fine-grained measurements of network flows. Existing works present solutions for traffic monitoring/measurement with different aims such as (a) flow measurement, (b) delay measurement (Round Trip Time), (c) packet-loss, (d) TM estimation, and (e) available bandwidth detection. These studies implemented their proposed solution with various techniques such as Single flow Statistic Request (SSR) known as per-flow polling, Polling All (PA) approach, which is also known as per-switch polling request, the combination of SSR with PA, and wildcarding rules and flow match. However, existing studies that either propose passive or active measurement methods can be broadly categorised into three main streams.

These approaches mostly focus on flow-based for timely-based monitoring and measurement in the network. This section presents a comprehensive overview of existing SDN monitoring and measurement approaches. Figure 2.7 depicts the classification of SDN monitoring and measurement approaches.

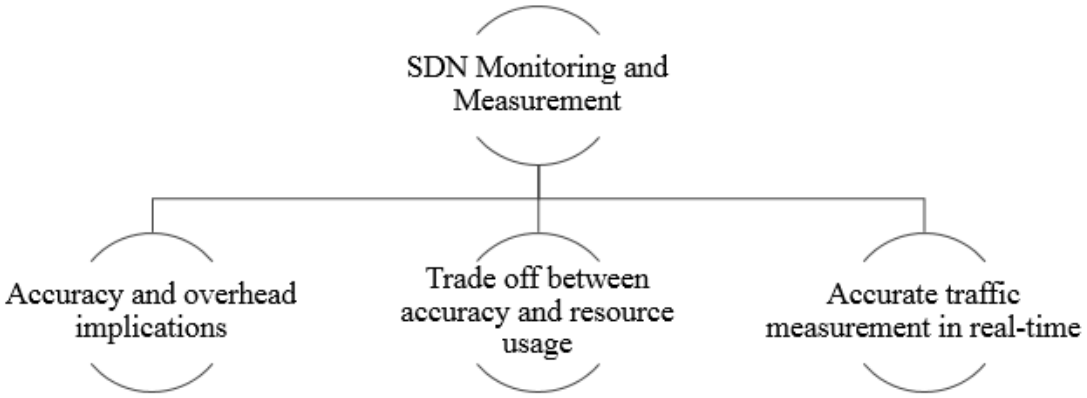


Figure 2.7: Classification of SDN Monitoring and Measurement Challenges

The first branch targets the trade-off between accuracy and the overhead implications of measurement approaches. Several techniques are used to overcome this problem, such as traffic sampling, aggregation, intelligent queries, etc. The second branch concentrates on finding a balance between accuracy and resource usage. Lastly, the third branch mainly focuses on providing accurate traffic measurement in real-time for reactive/proactive decision-making. To address the SDN monitoring/measurement challenges, existing studies proposes various methods and strategy to tackle their objectives. The strategies benefit from either an active or passive approach or a combination of both. Figure 2.8 depicts the strategies adopted in the existing proposed SDN monitoring/measurement methods.

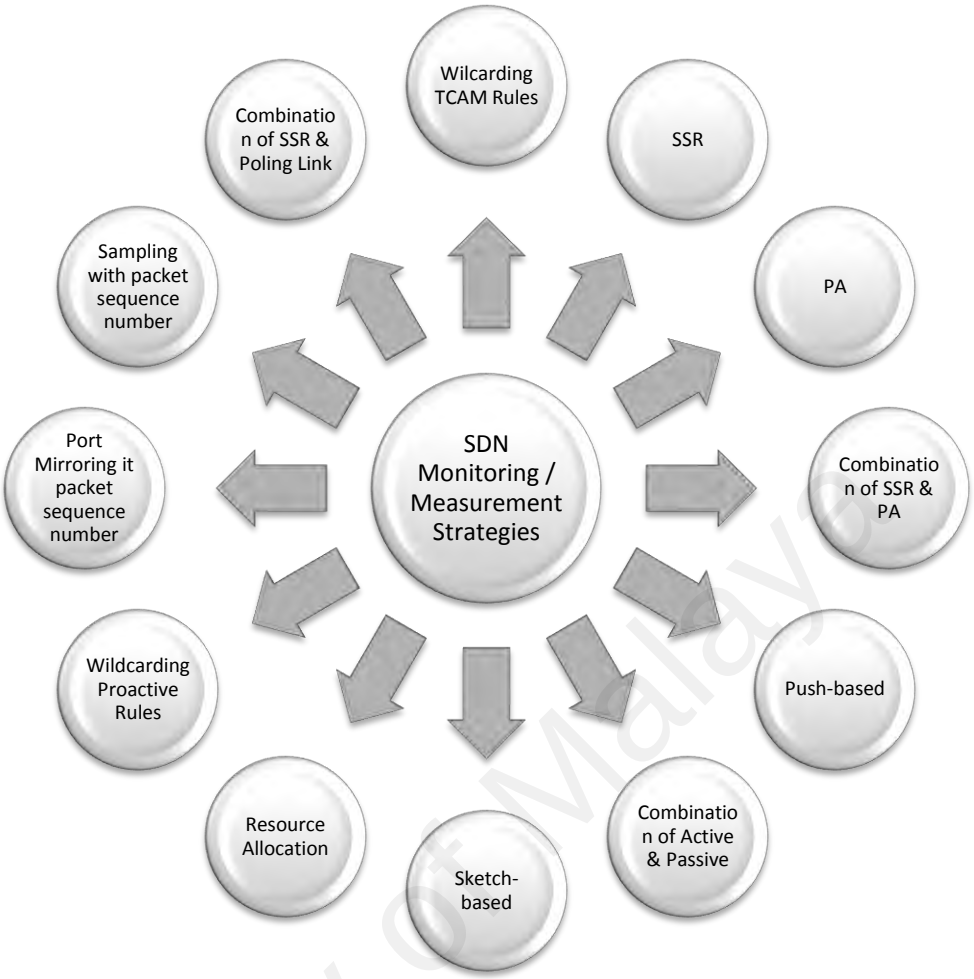


Figure 2.8: Strategies adopted in the exiting proposed SDN monitoring/Measurement methods

2.4.1 SDN Traffic Measurement Accuracy and overhead implications

Continuous monitoring of the network often introduces overhead, which needs to be taken into consideration as a trade-off with traffic measurement accuracy. To balance the accuracy of measurement and overhead implications, the proposed studies fall into five main categories, namely wildcarding TCAM rules, Single flow Statistic Request (SSR), Polling All (PA) approach, Push-based (passive) and a combination of SSR and PA.

2.4.1.1 Wildcarding TCAM rules

(Jose et al., 2011) studied the measurement of a large-scale traffic aggregation in commodity switches, by proposing a framework where switches match packets against a small collection of wildcard rules available in Ternary Content Addressable Memory

(TCAM). This approach significantly reduces the overhead of the controller, because the switch processing the packet identifies the matching rules locally and determines if it needs to drop the packet or forward it. The framework was deployed in several campus and backbone networks and evaluated using the Hierarchical Heavy Hitter (HHH) program to understand the trade-off between accuracy and overhead. However, as matching rules need to be frequently updated this causes extra overheads, which is a major issue in this approach when the traffic scale is large. Although new and existing rules require a separate controller, additional mechanisms need to be installed to discover and monitor HHHs. Also, the framework requires an extra controller to read the flow counters and install new rules. Furthermore, the framework measures the flows based on wildcarding the layer three network addresses (IP) which is insufficient for fine-grained measurement. iSTAMP (Malboubi et al., 2014) presents a fine-grained traffic flow measurement, which applies a (de)aggregation (wildcarding matching rules with the prefix keys) measurement mechanism. It dynamically partitions the TCAM entries to allow fine-grained or coarse-grained measurement tasks of incoming flows. For example, when direct per-flow measurement is required, TCAM is divided into two parts in which one partition is used for aggregated flow measurement and another partition for de-aggregation. The disaggregation mechanism is used for the so-called "most rewarding flows" (defined as flows with the highest impact on the ultimate monitoring application performance). Flows are "stamped" for direct measurement if they are deemed to be important. iSTAMP uses an intelligent Multi-Armed Bandit (MAB) based algorithm to process these two sets of measurements, which are then jointly processed to estimate the size of all network flows using different optimisation techniques. iSTAMP seems to make a good trade-off between the used measuring resources and accuracy, but it also faces several issues. First, the priority and wildcard-based matching strategy used by SDN switches implies that only the flows with a same prefix can be potentially aggregated by

one rule, but iSTAMP ignores the flow aggregation constraints, leading to infeasible aggregated measurements. Second, to find out the most rewarding flows, it uses all of the TCAM entries to measure all individual flows over multiple time intervals, which will introduce non-negligible measurement costs.

2.4.1.2 **Single-flow Statistic Request (SSR)**

OpenNetMon (Van Adrichem et al., 2014), defines if the end-to-end QoS parameters are actually met for each flow in ISPs. It is a pull-based active measurement approach where network flows are continuously monitored between predefined endpoints for throughput, packet loss and delay using polling edge-switches. For throughput, it only queries the last switch on the forwarding path with an adaptive frequency. The counter returns the number of packets(s) of each flow in the sampling interval (T), and the forwarding path throughput can be obtained as S/T. For packet loss, it polls the flow counters on the ingress and egress switches for a given flow and calculates the difference. For delay measurement, it uses the SDN controller to inject probe packets into the network along a given path and then loop them back to the controller. Thus, able to calculate the delay for the given path using the round-trip time between ingress and egress switches. OpenNetMon uses an adaptive fetching mechanism to pull data from switches where the rate of the queries increases when flow rates differ between samples and decreases when flows stabilize. Using pulling-based approach guarantees the accuracy of obtain results. However, OpenNetMon polls the switch for each and every active flow in the network for different purposes which can impose extra overhead on the network as well as the switches. OpenTM (Tootoonchian et al., 2010), proposes a traffic matrix estimation system using a simple strategy for querying flow table counters. It works based on keeping statistics for each active flow in the network. The information about active flows are kept by the controller and pulled from the switches periodically and then compared for accuracy. OpenTM is an active bull-based network-wide measurement

approach that at the end will introduce overhead in the process of periodically pulling statistical information from switches across the network. It introduces five selection algorithms to choose the switch for polling. (1) querying the last switch (2) querying switches on the flow path uniformly random (3) round-robin query (4) non-uniform random querying that tends to query switches closer to the destination with a higher probability (5) Querying the least loaded switch. OpenTM showed that querying switches closer to the destination with a higher probability has a better performance compared to other switch selection strategies. However, it demonstrated that querying the last switches results in the most accurate TM but it imposes substantial load on the edge switches. Furthermore, OpenTM uses a combination of selection methods to select switches for pulling information; this may lead to some measurement inaccuracy as investigated.

2.4.1.3 **Combination of Active and Passive**

Payless (Chowdhury et al., 2014), an active monitoring framework for SDN which implements pull and push-based approach at the same time. It focuses on the tradeoff between accuracy and message interaction overhead. It provides a flexible RESTful API for flow statistics collection at different aggregation levels. As the frequency of polling the switches determines monitoring accuracy and network overhead, Payless is that it uses an adaptive statistics collection algorithm to attain accurate information in real-time without incurring significant network overhead. The adaptive algorithm is an elastic polling regulator by which it set a higher polling frequency for flows that significantly contribute to link utilisation, and it set a lower polling frequency for flows that do not significantly contribute towards link utilisation at that moment. This, it polls the switch more frequent when gets bigger and less frequent when flow does have a significant change. Payless, uses Floodlight controller's API to implement the proposed mechanism. It has been shown through evaluation that Payless can indeed provide low overhead and

can achieve higher accuracy of statistics collection. But, the accuracy seems to increase only at the expense of the overhead.

2.4.1.4 **Push-based (Passive Measurement)**

FlowSense (C. Yu et al., 2013) uses passive measurement (push-based) to eliminate the extra overhead imposed by the pull-based measurement approach in which the network sends statistics about the flows instead of querying the switches on demand. The approach utilises packetin and FlowRemoved messages, which are sent by switches to the controller when a new flow comes in or upon the expiration of a flow entry (time-out event). The FlowRemoved message is sent from switch(es) whenever a flow time-out is reached. The FlowRemoved message contains statistical information about the flow, such as packet/byte-count, matches etc. These capabilities are provided by OpenFlow to query switches for the number of packets or bytes in flows matching a specific rule or traversing a network link. Evaluation results show that FlowSense has a promising performance compared to other approaches and can accomplish 90% of link utilisation in less than three seconds. However, if the flow entry has a long time out, the propose method is unable to generate a timely-based measurement related to the flow. Another drawback is that the flows with proactive rule cannot be triggered for time-out events. Thus, proactive flows cannot be measured by FlowSense. Furthermore, as wildcard rules limit the number of FlowRemoved, many flows cannot be triggered if wildcarding is adopted. Rules that are proactively installed by the operator without controller intervention are proactive rules. These rules may have unlimited time-out.

2.4.1.5 **Combination of SSR and PA**

FlowCover (Su et al., 2014), presents a low-cost flow monitoring scheme aimed at reducing the network communication overhead imposed by the polling flow for timely-based measurement. FlowCover pulls flow information from the switches using SSR strategy for the new arriving flows and utiliaes PA for polling the aggregated existing

flows in a switch. An algorithm chooses the most cost-effective switches and removes the covered flows, until all flows are covered. Later CeMon (Su et al., 2015), presented an adaptive fine-grained polling scheme to avoiding unnecessary polling, which increases the polling frequency by a variable if the difference in byte-count from two previous intervals exceeds a predefined threshold. However, FlowCover and CeMon may result in some degree of inaccuracy if the algorithm misses the SDN switches to poll.

A study by (Megyesi et al., 2016) presents a bandwidth measurement system using active measurement that applies the pull-based approach. The method polls the SDN switches to attain flow statistics, and calculates the available bandwidth by subtracting the path capacity from the total byte-count pertaining to the mentioned path. (Megyesi et al., 2017) introduce a local timestamping mechanism to increase the accuracy of the measurements by preventing measurement errors imposed by network jitter. However, the proposed timestamping mechanism is an extra feature that is not a part of standard OpenFlow features and unavailable in the commodity switches.

The authors in (Hongli Xu et al., 2017) and (H. Xu et al., 2017) argue that implementing the PA approach may degrade the switch CPU utilisation. The study presented a wildcard-based traffic flow measurement method with the aim of reducing the bandwidth costs and processing delay associated with the polling switches. The method applies an approximation algorithm to select the SDN switches, and polls the flows that can be covered using wildcarding. In addition, it uses SSR for the rest of flows that cannot be covered by wildcarding. The proposed method achieved a meaningful reduction in bandwidth overhead and processing delay. However, the proposed method lacks the ability to measure fine-grained flow specifications, as wildcard rules can only

be applied on the IP source/destination and input port of the SDN switches (Pfaff et al., 2012).

(Yang & Yeung, 2017) propose a weighted switch selector algorithm to weight the switches that are divided into two groups, “with and without lonely flows”. “Lonely flow” is a flow that can only be found in one switch and can be polled with the SSR approach. In practice, this can represent the local traffic among different IP subnets connecting to the same edge switch/router. The switches with lonely flows will be considered first. Afterwards, the controller polls the switches containing not-yet-covered flows, using the PA approach. Therefore, since the “lonely flow” only passes from a single switch, the statistic is not counted in the PA as it has already been covered, thus saving polling and reducing network communication overhead. However, the method keeps a complex algorithm running for each and every active flow in the network that might impose extra overhead in the controller and a delay to identify “lonely flow” if there is a large scale of flows in the network.

Table 2.4 summarises the discussed approaches of traffic measurement in SDN for the trade-off between accuracy and overhead implications.

Table 2.4: Current traffic measurement method in SDN for the tradeoff between accuracy and overhead implications

No	Type	Solution	Objective	Approach	Target Network
1	Active	Jose et al	Reducing overhead for flow measurement	Pull-based	Campus and ISPs
2	Active	OpenNetMon	Accuracy in measuring throughout, packet loss, and delay	Pull-based	ISPs
3	Active	iSTAMP	Reducing overhead for fine-grained flow measurement.	Pull-based	Backbone Network
4	Active	OpenTM	Accurate Traffic Matrix estimation	Pull-based	A synthetic topology

Table 2.4, continued

5	Active & Passive	Payless	Reducing message interaction overhead and increasing accuracy of traffic flow measurement	Pull & push-based	A synthetic topology
6	Active	FlowSense	Eliminate extra overhead of measuring link utilisation	Pull-based	Campus Datacenter
7	Active	FlowCover	Reducing network communication cost for measuring traffic flow	Pull-based	Campus Datacenter
8	Active	CeMon	Reducing network communication cost for measuring traffic flow with an adaptive fine-grained polling	Pull-based	Campus Datacenter
9	Active	(Megyesi et al., 2016)	Accuracy in available bandwidth measurement	Pull-based	Synthetic topology
10	Active	(Megyesi et al., 2017)	Accuracy in available bandwidth measurement with a local timestamping mechanism	Pull-based	Synthetic topology
11	Active	(Hongli Xu et al., 2017)	Reducing switch CPU overhead with partially wildcarding	Pull-based	Campus Datacenter
12	Active	(H. Xu et al., 2017)	Reducing switch CPU overhead with approximation algorithm	Pull-based	Campus Datacenter
13	Active	(Yang & Yeung, 2017)	Reducing communication overhead of flow measurement	Pull-based	Synthesis topology

2.4.2 SDN Traffic Measurement Accuracy and resources usage

Obtaining accuracy for monitoring/measurement and traffic estimation requires CPU and memory constraint in the switch, which is a challenging problem (Liu et al., 2016). The above measurement techniques were not concerned with resource usage and its effect on measurement accuracy. This section presents the proposed works concerning resource usage (network device overhead) and accuracy.

2.4.2.1 **Sketch-based approach**

Sketches can be used for many measurement tasks such as heavy-hitter detection (Bandi et al., 2007), traffic change detection (Schweller et al., 2004), flow size distribution estimation (Kumar et al., 2004), global iceberg detection (Huang et al., 2009), and fine-grained delay measurement (Schweller et al., 2004). Sketches are essentially compact data structures used in streaming algorithms (Bar-Yossef et al., 2002) to store summary information about the state of packets. Compared to flow-based counters, sketches have two key properties: (a) low memory usage: The size of summary information (sketch outputs) is significantly smaller than the input size. For example, the bitmap (Estan et al., 2003) is a simple sketch that maintains an array of bits to count the number of unique elements (e.g., IP source addresses); (b) provable trade-offs of memory and accuracy: Sketches often provide a provable trade-off between memory and accuracy, although the definition of accuracy depends on the actual sketch function (Cormode & Muthukrishnan, 2005).

In an effort to find a happy zone between accuracy and resource usage using a sketch-based approach, OpenSketch (M. Yu et al., 2013), a software-defined measurement architecture, is proposed as an alternative to OpenFlow. OpenSketch uses a measurement library in the control plane to automatically configure and manage resources for measurement activities. The library makes it easier to customise and apply theoretical algorithms to measure flows in commodity switch components. OpenSketch can be used for several measurement activities including HHH measurement, traffic distribution and link utilisation. The main barrier for OpenSketch as a marketable SDN traffic measurement solution is the need for upgrading network nodes, which is a very expensive undertaking. Furthermore, it is very rigorous and time consuming to standardise a new protocol. OpenFlow has already taken off, is widely accepted as an industry standard in datacentre environments and is increasingly implemented in commodity switches. With

OpenFlow gaining momentum, it will be adopted faster by ISPs and research communities. In a similar vein, JOTA (Su et al., 2017) proposed a joint optimisation algorithm to improve the performance of the task assignment by considering both the estimation of resource usage in the application layer and the available resource in the task assignment layer, it then calculates the task assignment problem as a mixed integer nonlinear programming problem. It decomposes the nonlinear programming algorithm into the resource compression stage and the task assignment stage. A two-stage heuristic is proposed to efficiently produce the task assignment. Specifically, JOTA compresses the resource usage in the first stage and relaxes one constraint to transform the problem to a Multi-Resource Generalised Assignment Problem (MRGAP) in the second stage. It then leverages an approximation algorithm to assign measurement tasks. However, due to the sophisticated optimisations and assignment complexity, JOTA can only support assigning tasks to one switch, and like any other sketch method it suffers from high complexity.

2.4.2.2 **Resource allocation**

DREAM (Moshref et al., 2015) is a dynamic resource allocation measurement framework that balances between user-specified level of accuracy and resource usage for measurement activities. In DREAM, resources are not allocated before the execution of the measurement task, but are dynamically deployed to achieve the desired level of accuracy based on traffic characteristics. DREAM framework is tested using HHHs programs show that DREAM can support more concurrent tasks with higher accuracy than several other alternatives. Aligning measurement tasks between the host and the network is also a major activity that in the end may reduce the overhead of an active measurement approach. The work in (Dusi et al., 2014) argues that current controller applications in SDN systems are designed to be proactive, which may require the switches to accommodate a number of flow table entries that exceed the capabilities of their

TCAMs. While equipping SDN switches with more powerful TCAMs is a feasible option, this may come at the expense of increasing operation and power consumption cost. The study proposes that controllers should consume resources efficiently using a reactive logic control approach. As in DREAM, the study suggests that resources must be allocated and freed depending on the network load, the effective behavior of the flows, their granularity and their inter-packet arrival time. The evaluation of the system shows that such approach is promising to enhance the traffic measurement flexibilities without extending the flow tables. Another algorithm called Baatdaat (Tso & Pezaros, 2013) uses OpenFlow running on NetFPGA programmable switches, which permits real-time dynamic flow scheduling in datacenters. The proposed algorithm can adapt to instantaneous traffic bursts as well as to average link load by using spare DC network capacity to mitigate the performance degradation of heavily utilized links. Experiments show that Baatdaat can reduce network-wide maximum link utilisation by up to 18% equal cost multipath (ECMP).

Finally, the proposed HONE platform (Sun et al., 2015) presents a uniform stack for a diverse collection of measurements in SDN-based systems. HONE uses software agents residing on hosts, and a module interacting with network devices. Since continuously collecting statistical data about network flows is expensive, HONE offers two techniques to process flow statistics: The first technique, known as lazy materialisation of the measurement data, uses database-like tables for uniform abstract representation of statistical data collected from hosts and network devices. The aim of this technique is to minimize measurement overhead by allowing the controller and the host agents to analyze queries of necessary statistics for multiple management tasks at appropriate frequencies. The second technique offers data parallel streaming operators for programming the data-analysis logic. The operators can also be used in a hierarchically fashion for aggregate analysis among multiple hosts. Scalability is a main problem in deploying HONE as

software agents need to be installed in every host and then synchronized to populate the statistical tables in a timely fashion to process meaningful queries.

2.4.2.3 **Wildcarding proactive rules**

Flame (Han et al., 2016) is a flow level TM estimation to minimise the CPU and TCAM utilisation on the switch. The authors argue that polling every single flow for constructing the TM is unsalable due to the limitation in switch TCAM. Flame contains a traffic estimation algorithm "Poisson Shot-Noise Process" to predict traffic behaviour changes in the near future. It adopts a proactive Flow Rule instantiation which wildcards the predicted future traffics and install them into the TCAM. The proposed method results in a reduction in TCAM usage, however, due to implementing the wildcard approach, it is insufficient for fine-grained measurement. Furthermore, timely-base TM estimation is restricted to ten seconds.

Table 2.5 summarises the discussed approaches of traffic measurement in SDN for the trade-off between accuracy and resource usage.

Table 2.5: Current traffic measurement method in SDN for the tradeoff between accuracy and resource usage

No	Type	Solution	Objective	Approach	Target Network
1	Active	OpenSketch	A generic and Accurate monitoring framework that support several measurement activities.	Sketch-based	Synthesis topology
2	Active	JOTA	Improving the performance of the task assignment	Sketch-based	Synthesis topology
3	Active	Dream	Accurate and low-cost resource allocation and Higher accuracy for concurrent tasks	TCAM-Based	Synthesis topology

Table 2.5, continued

4	Active	(Dusi et al., 2014)	Reducing cost of resource in measurement tasks	TCAM-Based & Proactive rule installation	Synthesis topology
5	Active	Baatdaat	Low-cost Scheduling algorithm for real-time for direct measurement	A hardware-assisted traffic monitor that measures link utilisation on the switches at line-speed	Synthesis topology
6	Active	HONE	minimize measurement overhead imposed by multiple measurement task	software agents residing on end-hosts	Synthesis topology
7	Active	Flame	Minimize the CPU and TCAM utilisation for flow level TM estimation	TCAM-Based	University datacenter
8	Passive	PLANCK	Accurate and real-time traffic measurement	port mirroring & packet sequence number	Fat-tree datacenter

2.4.3 SDN Traffic Measurement Accuracy in Real-time

Traffic measurement in SDN relies extensively on collecting statistical data about network flows in real-time. With the increasing popularity of real-time services such as voice and video, network monitoring has become a significant task in network operation. The large amount of detailed statistics provided by the hosts may raise a scalability issue for real-time analysis, specifically when the measured data are used for time sensitive applications.

2.4.3.1 Port Mirroring with packet sequence number

(Rasley et al., 2015) propose PLANCK, a software-defined measurement architecture, which utilises the capability of port mirroring that exists in most commodity switches.

The extracted measurement statistics of the network flow is achieved in 280 microseconds to 7 milliseconds on a 1 Gbps commodity switch and 275 microseconds to 4 milliseconds on a 10 Gbps commodity switch. This is faster than in traditional networks by orders of magnitude. Planck exploits the port mirror feature of switches. With this feature, network packets on other ports are copied to a specified port connected to a collector. The collector uses the TCP sequence numbers to estimate the flow rates. By collecting two distinct copy packets of the same flow, the collector can compute the flow rate. Note that Planck uses protocol-specific information (e.g. the TCP sequence number), which as such is not suitable for flows that do not have sequence numbers. Thus, in comparison with the proposed mechanism, Planck requires extra hardware resources and only works for traffic for specific protocols.

2.4.3.2 **Sampling with packet sequence number**

Using sampling-based SDN measurement methods, IBM research lab proposes OpenSample (Suh et al., 2014), which leverages sFlow (Phaal & Lavine, 2004) packets to provide near–real-time measurements of both network load and individual flows by capturing packets from the network. OpenSample is a low-latency platform that uses TCP sequence numbers from the captured headers to measure accurate flow statistics. Using sampling for network monitoring allows OpenSample to have a 100-millisecond control loop rather than the 1–5 second control loop of traditional polling-based approaches. It is implemented with Floodlight OpenFlow controller and evaluated on a testbed comprised of commodity switches. One of the main advantages of OpenSample is that it considers any TCP flow, hence it can detect elephant flows (large and continuous flows), and requires no modification to switches, making it highly marketable.

2.4.3.3 **Combination of SSR and Poling Link**

(Chen et al., 2016) argue that there is a serious scalability issue for the Plank collector: it reserves a single port on each switch for port mirroring, and reserved ports are

connected to the collector to analyse the traffic; hence, it is unclear how Planck can scale to support large data centres. In addition, for OpenSample and Planck, they are unable to detect traffic flow that does not have a sequence number in a packet. The author propose a scalable load-aware low-latency two-tier measurement and control platform to detect real-time congestion in the network. The first tier provides a scalable low-delay congestion detection mechanism that can be used for large SDNs without additional costs, using pulling port statistics. It provides a macro view of the network, that is the utilisation of all links in the network by collecting packet-level link statistics. The second tier uses flow-level statistics to deal with upcoming congestion. (Gangwal et al., 2017) also adopt such as two-tier strategy for real-time link loss and link delay. However, none of the frameworks are able to capture fine-grained flow traffic measurement, as polling port statistics only provide packet and byte counts of the specific port (link) rather than a flow.

Table 2.6: Current traffic measurement method in SDN for accuracy in real-time

No	Type	Solution	Objective	Approach	Target Network
22	Passive	OpenSample	Accurate Low-latency traffic measurement	Sampling & Packet sequence number	Synthesis topology
23	Active	(Chen et al., 2016)	Accurate and Low-latency congestion detection	SSR & Polling Link	Synthesis topology
24	Active	(Gangwal et al., 2017)	Accurate and for real-time link loss and link delay	SSR & Polling Link	Synthesis topology

Table 2.6 summarises the discussed approaches of traffic measurement in SDN for accuracy in real-time. It is worth mentioning here that several works such as network monitoring, fault tolerance, and topology update, although beyond the scope of this thesis because they are not specifically related to traffic measurement, are equally important for

the wider sense of network traffic SDN domain, and can be studied in (Hu et al., 2014) and (Akyildiz et al., 2014).

2.5 Summary

This chapter presented a broad overview of network traffic monitoring and measurement implications. Section 2.1 studied various aspects of traditional traffic monitoring and measurement approaches, ranging from the capabilities of network devices such as SNMP to packet/flow monitoring and sampling. It was concluded from the literature reviews that traditional approaches for measuring network traffic are insufficient in terms of accuracy. Although some of the approaches, such as packet/flow sampling, solve the issues of overhead complications imposed by the active approach, they still require extra hardware/software agents and collectors for analytical and statistical calculation. In addition, the calculations mentioned, produce overhead in the agent and analyser hosts and still cannot be accurate as basically sampling is a probabilistic method to estimate the taken samples. In the second part in section 2.2, a comprehensive overview of SDN was presented to familiarise the readers with the flexibility of this emerging technology, elaborating on different layers of SDN architecture and functionality. The section also demonstrated some of the well-known control planes that are used for academic and commercial purposes. Section 2.3 presented a light-weight background of SDN measurement and the detailed native approaches offered by OpenFlow that perform monitoring/measurement. In section 2.4, a comprehensive overview of state-of-the-art SDN measurement solutions was presented. From the literature, it can be comprehended that next generation networks require timely-based measurements that continuously measure traffic for any unexpected change in the network. Network applications demand accurate and fine-grained measurement frameworks that improve QoS constraints. However, accuracy in measuring flow level network traffic comes with costs, such as resource usage and network overheads imposed

by the pull-based approach, which should be taken into consideration for scalability issues in a large-scale network. In addition, many delay-sensitive applications in the network demand low-latency measurement and real-time statistical monitoring about network flows. As can be discovered from the literature, SDN measurement approaches lack a comprehensive design that combines the benefits from different strategies and tackles their challenges. However, exploiting the advantage of some strategies may disrupt the performance of other strategies. For example, adopting a pull-based strategy can offer the most accurate and real-time flow statistic measurement result, but at the expense of overhead in the network and high usage of resources. Therefore, one major objective that can be concluded from reviewing state-of-the-art approaches and analysing their challenges, is to design a framework that is able to deal with the accuracy and multi-objective cost with low latency results (in real-time). The next chapter clarifies the problem and formulates the aforementioned challenge; it also presents experimental proof.

CHAPTER 3: PROBLEM FORMULATION

The measurement task imposes overheads in different aspects that affect the control plane and the data plane at the same time. The severity of the overheads may vary in different network deployment models. This chapter aims to analyse the problem that was highlighted in Chapter 1. It also conducts an in-depth investigation to show the impact of flow measurement on different aspects of overheads. In addition, different network deployment models such as out-of-band and in-band are formulated. This study focuses on three imposed overheads, from which two were highlighted in the literature review, namely communication and message interaction overheads imposed by pulling flow statistics. It then introduces a new overhead that is generated by reading and calculating the measured flows' statistics.

The rest of the chapter is organised as follows. Section 3.1, introduces a preliminary notation to formulate the problem. Section 3.2 explains the problem analysis in different network deployment models for various aspects of overheads. This is followed by an elaboration on the problem of synchronisation of multiple controllers in section 3.3. Finally, section 3.4 shows a light-weight experiment for clarification of overhead aspects imposed by different approaches for pulling flows.

3.1 Problem Definition

This section clarifies the problems and formulates them in out-of-band and in-band network deployments. The formulation in this study is explained in a directed graph $G = (V, E)$ in which V devotes a set of switches $\mathrm{V} = \{v_1, v_2, \dots, v_n\}$ and E represents the set of links between switches and $\mathrm{f} = \{\mathrm{F} : f_i \in \mathrm{F},\ 0 < i < m\}$ represents a subset of flows where $\mathrm{F} = \{f_1, f_2, \dots, f_m\}$ is the total current flows (universe) in the network and $C = \{c_1, c_2, \dots, c_k\}$ a set of SDN controllers. Table 3.1 lists the notations of the formulations used throughout thesis.

Table 3.1: Notation of problem formulation

Notation	Description
$G = (V, E)$	The target network with V as a set of switches and E as a set of links.
F	Universe flows in the network.
C	A set of Controllers.
n	Message (request/reply/controller-to-coordinator)
l_{rq}	The Length of request message (122 bytes).
l_{rp}	The length of reply message (82 bytes).
l_{sf}	The length of every entry in a file (136 bytes).
l_{udp}	Minimum length of UDP message (header and body 60 bytes).
θ	The number of instruction taken to fragment stat-reply file.
λ	The number of instruction taken to read stat-entry.
k	Coordinator
$h_{\alpha\beta}$	The number of hops (nodes) from switch v_α to controller c_β

Traffic flow measurement imposes various overheads that are associated with measuring flows and collecting their statistics. These overheads have multiple aspects that affect the performance of the overall SDN network. The measurement task can affect (1) the SDN control network, i.e. links that connect switches to the controllers, and (2) the logically centralised controller.

The effect of flow measurement in the SDN control network is referred to as communication overhead, which determines the size of traffic imposed by generating flow stat-request messages and stat-reply messages in bytes. Furthermore, the logically centralised controller can also be affected by the calculation of raw data. The raw data are the flow statistics processed by the controller. This aspect has severe impact on the functionality of the centralised controller if the raw data volume is too high. This overhead is the so-called controller overhead, which determines the CPU utilisation of the controller that has been allocated for the calculation of flow statistics. Last but not least is the message interaction imposed by sending the flow stat-request message and receiving the reply message. Although the message interaction overhead impacts the SDN

centralised controller, it is still considered as an individual measurement overhead, which is different from the controller overhead. Unlike the controller overhead that shows the utilisation of the controller's CPU, the message interaction overhead turns the SDN controller into a bottleneck for generating and sending packetout (stat-request) messages and receiving packetin (stat-reply) messages from the switches.

As mentioned in the literature review, there are two native approaches in OpenFlow for collecting timely-based flow statistics, namely (1) single stat-request (SSR), (2) wildcarding all fields to collect all flows (polling all).

In order to collect flow statistics, SSR generates two messages in each time interval. The number of the generated message is increased to four messages for two flows and this increment continues in twice the number of flows. The main drawback of this approach is the imposed overhead for generating a huge number of request and reply message in the network, which also utilises the CPU cycle of network devices as well as the SDN controller.

PA collects all the active flow statistics in a switch with only two messages, one request and one reply message. This can significantly reduce the costs of message interaction and communication, as well as the repeated reply headers for a high number of flows. However, excessive use of the second approach causes flow statistics to overlap, which imposes extra message interaction overheads and communication costs as well as an overhead in the controller. Another drawback of PA is the lack of control on flow queries, as it pulls all the active flows in the network regardless of actual need.

3.2 Overhead Aspect (Metrics)

The overheads mentioned above, i.e. communication, controller and message interaction overheads in out-of-band and in-band network deployment, are formulated as follows:

3.2.1 Communication overhead

According to the OpenFlow specification 1.3 (Pfaff et al., 2012), the minimum length of flow stat-request message, l_{rq} in wire is 122 bytes (Su et al., 2015). More specifically, 14 bytes for ethernet header (layer 2 header size), 52 bytes for IP and TCP headers (20 bytes IP and 32 bytes TCP header size) and 56 bytes for OpenFlow statistic (16 bytes OpenFlow multipart message and 40 bytes OpenFlow match size). However, this length increases by setting extra specifications of flow such as various fields in match. Table 3.1 explains the request message structure and size for a single flow. Figures 1 and 2 in Appendix A depict the structure of the multipart message and the flow stat-request message, respectively.

As can be seen in Table 3.2, the total length of empty flow stat-request is 122 bytes. However, the match field is variable, and its length is extended according to a set of matching fields. This study follows the flow match field from Cisco to define a network flow as 5-tuple matching fields such as (1) IP protocol, (2) source and IP address, (3) destination IP address, (4) source port address, and (5) destination port address. However, the field "ethernet type" is a requirement to define IP protocol as a matching field in OpenFlow (Pfaff et al., 2012). Therefore, similar to OpenTM (Tootoonchian et al., 2010), this study defines "ethernet type" as a matching field. Figure 3 in Appendix A shows the required fields for match fields with the description and length in bits. The total request packet length in the SSR approach with the defined match fields above captured by

Wireshark is 162 bytes, whereas, the length of a request packet in the PA approach is 122 bytes as there is no defined field to match.

Table 3.2: Request Message Structure and Size for a Single Flow

Specification		Size (byte)
Header	Ethernet	14
	IP	20
	TCP	32
OpenFlow Payload	Multipart request	16
	Flow static request (variable match size)	40
Total		122

The reply message includes a header whose length l_{rp} for SSR and PA is 82 bytes and 162 bytes respectively. The length for each single flow entry l_{sf} stat is 144 bytes. However, the reply message may split into multipart messages if the total size exceeds 64Kb, as the maximum size of a TCP packet in medium is 64Kb. In this case another multipart reply message is created as a reply message that contains the remaining flow statistics that could not be carried by a single reply message. Table 3.3 elaborates on the reply message structure and length. Figures 4 and 5 capture the structure of the multipart reply message and the stat-reply message for SSR in Appendix A. The length of the UDP message (Zander et al., 2007) containing the aggregated *statistic* sent by every local controller to the coordinator is 60 bytes, which is donated by l_{udp}. Therefore, the total communication overhead $Cost_{com}(\mathrm{f})$ of SSR and PA for polling a set of arbitrary flows f for each individual controller in out-of-band deployment is a linear function of f as in equation 3-1 and 3-2, respectively. The communication cost for SSR in out-of-band network deployment comprises the length of generated messages in every time interval that it equals the combination of the length of the request message and the header of reply message and the length of UDP message multiplied by the number of controller as it is shown in the equation 3-1. The communication cost of SSR in in-band network

deployment is calculated similar to the out-of-band deployment multiplied by the number of hops from each switch to its corresponding controller. Therefore, SSR generates minimum 2 messages for each flow at every time interval. However, this cost for PA in out-of-band network deployment comprises the combination of length of the request messages and the header of reply message and the length of all the requested flow entries in every time interval. Whereas, the generated total length is multiplied by the number of hopes in in-band network deployment. Equations 3-3 and 4-4 describe the linear formulation formula of communication overhead $Cost_{com}(f)$ of SSR and PA in in-band deployment. A UDP packet may contain empty datagram (no data). However, the minimum length in the wire over ethernet is 60 bytes. The minimum elements that contribute to the length of an UDP packet are *ethernet header*, *IPv4_header*, and *UDP_header,* which equals 14+20+8 = 42 bytes. However, as per by Linux host driver, extra bytes are padded to the packet to fulfil the ethernet requirement of minimum packet length.

Table 3.3: Reply Message Structure and Length

Specification		Size (byte)	
Header	Ethernet	14	
	IP	20	
	TCP	32	
OpenFlow Payload	Multipart request	16	
	Flow Stat Entry List	SSR	PA
		0	80
	Single Flow Statistic	144	
Total		226	306

(a) Out-of-band

$$Cost_{com}(\mathrm{f}) \cong \sum_{f_i} (l_{rq\,i} + l_{rp\,i} + l_{sf\,i} + (l_{udp} \times v_{kc})), \quad \forall f_i \in f\ (0 < i < m), f \subseteq F, v_{kc} \in v, v \subseteq V \tag{3-1}$$

$$Cost_{com}(\mathrm{f}) \cong (l_{rq} + l_{rp} + \sum_{f_z \in F} l_{sf_z})\, v + (l_{udp} \times v_{kc}), \quad \forall f_z \in F\ (0 \le z \le |F|), v_{kc} \in v, v \subseteq V \tag{3-2}$$

(b) In-band

$$Cost_{com}(\mathrm{f}) \cong \sum_{f_i} (l_{rq\,i} + l_{rp\,i} + l_{sf\,i} + (l_{udp} \times v_{kc})) \times h_{\alpha\beta}, \forall f_i \in f\ (0 < i < m), f \subseteq F, \quad v_{kc} \in v, v \subseteq V \tag{3-3}$$

$$Cost_{com}(\mathrm{f}) \cong (l_{rq} + l_{rp} + \sum_{f_z \in F} l_{sf_z})\, v + (l_{udp} \times v_{kc}) \times h_{\alpha\beta}, \quad \forall f_z \in F\ (0 \le z \le |F|), \quad v_{kc} \in v, v \subseteq V \tag{3-4}$$

3.2.2 **Message Interaction Overhead**

As the number of flows in the network increases, polling their statistic counters requires a symmetric growth with regard to the flow number. In other words, the more flows there are in the network, the more messages are interacted between the controller and switch(es). This makes the pull-based approach inefficient for continuous measurement with high-granularity due to consuming too much of the switch-controller's bandwidth as well as switch CPU (Mogul et al., 2010). Moreover, Sünnen (Sünnen, 2011) showed that when read-stats messages are sent too often, the switch's CPU utilisation and the number of spending messages increases. Thereby, given the network graph *G* with set of flow *f* and a set of controllers C, the total number of message interaction for each individual controller in SSR and PA can be found in a linear function of $Cost_{message}(f)$ in equations 3-5 and 3-6, respectively, where n_{rq}, n_{rp} and n_{udp} are

"ofp_flow_stats_request", "ofp_flow_stats_reply", and "udp for coordinator" messages, respectively. Therefore, the total message interaction of SSR comprises the multiplication of the number of requested flow and the combination of the total number of request message and reply message and UDP message. Whereas, this overhead for PA equals the multiplication of the number of switches and the combination of the total number of request message and reply message and UDP message.

$$Cost_{message}(f) = \sum_{f_i} (n_{rq_i} + n_{rp_i} + n_{udp_i}), \quad \forall f_i \in f\ (0 < i < m), f \subseteq F \tag{3-5}$$

$$Cost_{message}(f) = \sum_{v_j} \left((n_{rq_j} + n_{rp_j} + n_{udp_j} \right), \ v_j \in v, v \subseteq V \tag{3-6}$$

Subject to:

$$\rho = \{m \mid m: N\}, p = \begin{cases} 1, & Cost_{com}(\mathrm{f}) \leq 65536bytes \\ m \times 2, & Cost_{com}(\mathrm{f}) > 65536bytes \end{cases} \tag{3-7}$$

where the condition stated in equation 3-7 defines a reply message is split into two parts (two messages) if the length is greater than 65536bytes. The default packet size in the network cannot be greater than 64Kb for efficient transfer of data in the network.

3.2.3 Controller Overhead

The controller's overhead implies the utilisation of the controller's CPU, which is also referred to as the computation overhead (Megyesi et al., 2017). It can be defined by the numbers of instructions imposed by the execution, calculation and comparison of raw data to process byte-count i.e., calculating the byte-count of each flow and subtracting it from the previous count. The performance and throughput of CPU is measured from different perspectives such as Cycles Per Instruction (CPI), Million Instruction Per

Second (MIPS), and Transaction Per Second (TPS). One cycle is the minimum time it takes the CPU to execute any work. The clock cycle time or clock period is the length of a cycle. The clock rate, or frequency, is the reciprocal of the cycle time. Thereby, the CPU instruction rate is calculated by dividing the observed CPU cycle speed by the observed CPI (Zhang et al., 2013). However, determining the exact number of instructions applied by the CPU requires obtaining job information, i.e. calculation of statistic replies, comprising multiple tasks, each of which consists of multiple threads, which is out of the scope of this thesis. In addition, different CPU generations such as 32 or 64bit CPU registers perform differently with various CPU instructions such as Instruction Fetch(IF), Instruction Decoder(ID), Execution(EXE), Memory/IO(MEM), Write-Back(WB) each of which consists of various instructions that increase the clock cycle time (Yi & Ding, 2009). Therefore, analysing and calculating CPU performance by number of instructions is not rational in practice.

A simple criterion to observe the imposed overhead is to presume a constant value θ and λ indicating the number of instructions taken by the CPU for the fragmentation of the stat-reply files (data transfer) and reading stat-entry (arithmetic, data transfer, logical, conditional, and jump), respectively. According to (David Patterson, 2014), θ (reading stat-file and put into memory) and λ (subtracting current flow count from the previous one and put into memory) are equal to 2 and 6 instructions respectively. Table 3.4 shows the MIPS assembly instruction language used by the CPU. Therefore, the controller overhead $Cost_{controller}(f)$ in SSR, and PA analysing n-specific flows from set f in each interval is formulated as a linear function of m in equation 3-8 and 3-9 respectively.

$$Cost_{controller}(f) = \sum_{f_i}(3\theta_i + \lambda_i), \qquad \forall f_i \in f\ (0 < i < m), f \subseteq F \tag{3-8}$$

$$Cost_{controller}(f) = \sum_{v_j} (3\theta_j + \lambda_j), \qquad v_j \in v, v \subseteq V \tag{3-9}$$

Table 3.4: MIPS assembly instruction language taken by CPU. Adopted from (David Patterson, 2014)

$\boldsymbol{\lambda}$	$\boldsymbol{\theta}$
2 × Unconditional jump (Jump to target address, For switch, procedure return)	Unconditional jump (For procedure call)
Data transfer (Byte from memory to register)	Data transfer (Byte from memory to register)
Conditional branch (Compare less than)	
Arithmetic (subtract)	
Data transfer (Byte from register to memory)	

3.3 Synchronisation of Multiple Controller

Multiple controllers are required to share their collected flow statistics among each other in order to construct an integrated and understandable statistic of flows. For example, three controllers may be assigned to pulling a specific flow (i.e. TCP flows with dsc port 8080) from several switches in different sides of a network. Each controller receives the aforementioned flow statistic from their corresponding switches. Therefore, the total result of the measurement in each interval is the sum of all statistics collected by each controller in the same interval.

OpenFlow introduces a failover mechanism in specification version 1.2, in which multiple controllers can connect to a switch with different roles such as Master, Slave, and Equal. According to this mechanism, multiple controllers can request flow statistics from a switch, however, the switch responds to each controller with a reply message. For example, if there are four controllers and four switches, all controllers connect to all switches. Therefore, to pull a specific flow (i.e. TCP dsc port 8080), 32 requests and reply messages are generated in total, which can disrupt the functionality of switches due to

excessively pulling flows. In addition, all controllers pull the same flows, which results in overlapping of the statistic collection and causes a massive overhead in the controller for the calculation of identical raw data. Furthermore, the interaction of these messages imposes excessive network communication overhead. Therefore, due to the huge complexity of the mentioned OpenFlow failover mechanism, it is inapplicable for flow measurement purposes in a large network with many flows.

As mentioned earlier in Chapter 2, several papers propose different models of multiple-controller deployment and applied synchronisation (the east-west bridge) among the controllers. Each controller exchanges reachability and topology information between the inter-domain controllers (Xie et al., 2015). However, when using the east-west bridge, controllers can only share their network views, flow tables and status through complex steps such as starting a TCP handshake and constant "Hello" and "Echo" request/reply messages. Even if the east-west bridge approach can be applied on flow measurement synchronisation, it is still costly in terms of overhead imposed for inter-controller communication.

To the best of the researcher's knowledge, there is no existing study that focuses on the synchronisation of flow statistics on multiple SDN controllers.

3.4 Experimental Analysis

This section discusses findings from light-weight experiments investigating the impact of the pulling flow for measurement purposes. The experiments were conducted in Mininet (Mininet, 2015), and traffic was generated by D-ITG (Botta et al., 2012). Also, Wireshark was employed to collect and generate messages for measurement purposes. A linear topology was implemented, consisting of three switches of which each switch was connected to one host. The OpenFow controller is connected to the first switch and D-ITG server is connected to the last switch. Figure 3.1 depicts the topology of the

experiment. All the elements used in the experiment are comprehensively explained in the experimental setup in section 5.1.1. Figure 3.2 and 3.3 show the captured request and reply message for flow statistic collection, using SSR in out-of-band and in-band network deployment, respectively.

Figure 3.3 shows the number of packets by pulling one switch along all paths using SSR. It should be recalled that in out-of-band deployment switches are connected to the controller via a separate dedicated network, whereas in in-band configuration, switches connect to the controller through the same data plane network. Therefore, only two packets, one with 124 bytes and one with 218 bytes, are captured from all the available interfaces. However, the number of messages and their total length depend on the number of links directly connected to the switch interface in in-band deployment as shown in Figure 3.3. Therefore, if there are three switches along the path, the number of captured packets and their total length are multiple to the number of switches, which is three in this experiment. Figure 3.4 explains request and reply messages using the PA approach in out-of-band deployment. Section 3.2.1 and Figure 3 in Appendix A refer to the length of request and reply messages.

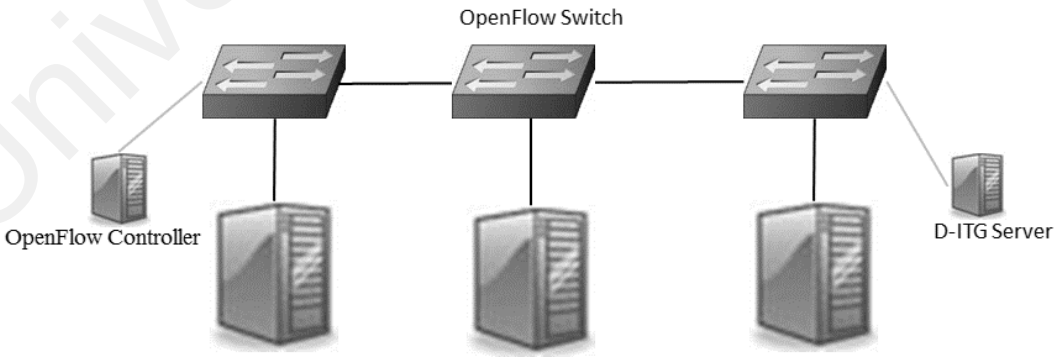


Figure 3.1: Request and reply message of SSR in out-of-band network deployment.

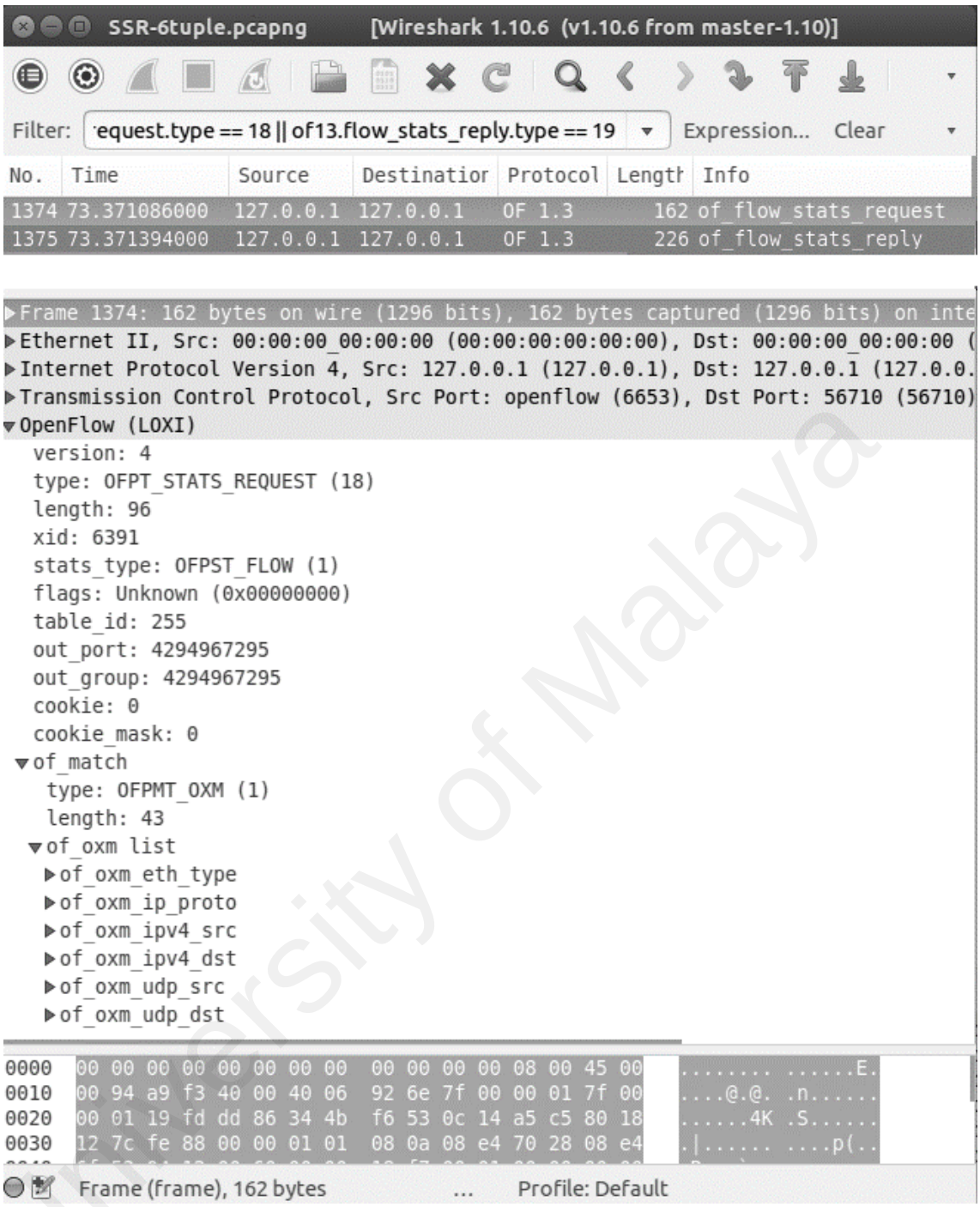


Figure 3.2: Request and reply message of SSR in out-of-band network deployment.

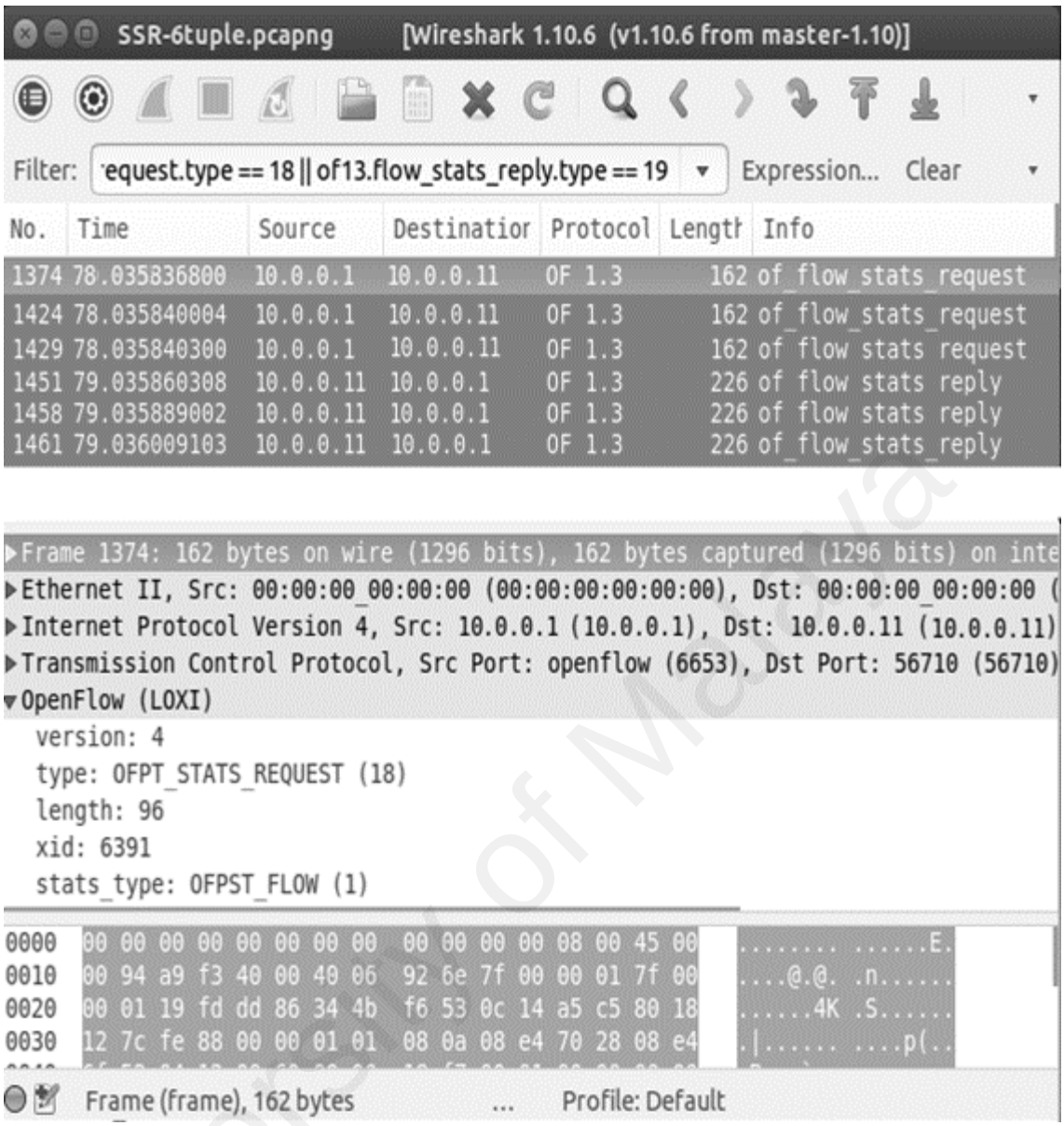


Figure 3.3: Request and reply message of SSR in-band network deployment.

Figure 3.4: Request and reply message in PA approach in out-of-band network deployment.

As can be seen in Figure 3.4, in the PA approach the request packet length is 122 bytes as no match field is set. A request packet with no match field simply means *all* (pulling all). In addition, the reply packet captured in Figure 3.5 reports the length of 306 bytes, from which it includes144 bytes for every flow statistic (of_flow_stat_entry in the figure), 80 bytes for flow_stat_entry list (referred to figure), 16 bytes for multipart message, and

66 bytes for packet header consists of ethernet, IP and TCP header (14+20+32). However, for every flow stat-entry (flows in the switch), 144 bytes are added to the packet length.

3.5 Summary

This chapter aimed to analyse the problem of pulling-based flow measurements and the resulting impact on different types of overheads. It proves the impact of applying SSR and PA in out-of-band and in-band network deployments for communication overhead. The chapter illustrated the formulation of different overheads imposed by implementing SSR and PA approaches in in-band and out-of-band network configurations. Additionally, the chapter experimentally disclosed the captured packets from OpenFlow approaches and showed the associated costs, i.e. communication overhead, with the packet length and number of flows. It also elaborated on the problem of multiple-controller synchronisation for flow measurement. The analysis was conducted in order to identify the behaviour of various OpenFlow native approaches with different flow numbers and network traffic patterns.

After disclosing and formulating the problem, the next stage is design and development. The next chapter discusses the proposed framework, where two designs are elaborated upon, namely for centralised and decentralised SDN controllers.

CHAPTER 4: MULTI-OBJECTIVE FLOW MEASUREMENT: FRAMEWORK

This chapter elaborates on the proposed multi-objective flow measurement method to effectively minimise various costs that are associated with the measurement of flows in a DCN, such as communication, message interaction, and controller overheads. The key objective of this chapter is three-fold: (i) discussing the proposed solution to minimise measurement costs, (ii) explaining the architecture of the proposed framework, (iii) presenting the proposed method for the decentralised (multiple) SDN controller. The remainder of this chapter is structured as follows:

Section 4.1 illustrates the proposed solution for the optimisation of different costs. It elaborates the OpenFlow group table feature, which is adopted for flow measurement purpose. It then introduces the fine-grained measurement specification feature for the proposed flow measurement method. A comprehensive description of the system framework is presented in section 4.2. Section 4.3 explains the building blocks of the proposed multiple-controller design. The section also explains the mathematical formulation for different costs caused by the proposed framework. In addition, the section shows the process flow of the proposed architecture. Finally, the chapter is summarised in section 4.4.

4.1 Proposed approach for optimisation of costs

This section presents a dynamic and multi-objective measurement approach that effectively minimises the costs associated with measuring flows. It abstracts all the demanded flows (flows to be measured) in the groups and sets the action of the groups to the corresponding flows' action. According to OpenFlow, flows are wildcarded either by all fields (PA) or "some cases bitmasked" such as *IP-scr*, *IP-desc* and input port (Pfaff et al., 2012).

The proposed approach adopts the *group table* introduced by (Pfaff et al., 2012) for traffic engineering, load balancing and fast-failover purposes. A group can either have a single or a list of action/bucket. There are four types of groups such as *All*, *SELECT*, *INDIRECT*, and *FAST-FAILOVER*, each of which has specific features. Interested readers are referred to (Pfaff et al., 2012) for detailed information. The proposed approach implements the *SELECT* type in which each packet entering the group is sent to a single bucket associated with its action. Thus, for a *group* in a switch, the proposed approach defines all the potential output actions related to the flows. In such a case, all the incoming flows or current flows are grouped without any intervention to the forwarding decision and central policy enforcement. Therefore, the action of a flow entry is set to the action or a list of actions for that group. The proposed approach then requests the group rather than pulling a single (SSR) or all flows (PA). Figure 4.1 shows the *SELECT* group type in OpenFlow 1.3. The pseudo code to construct group and mapping flows to the group is shown in Figure 4.2.

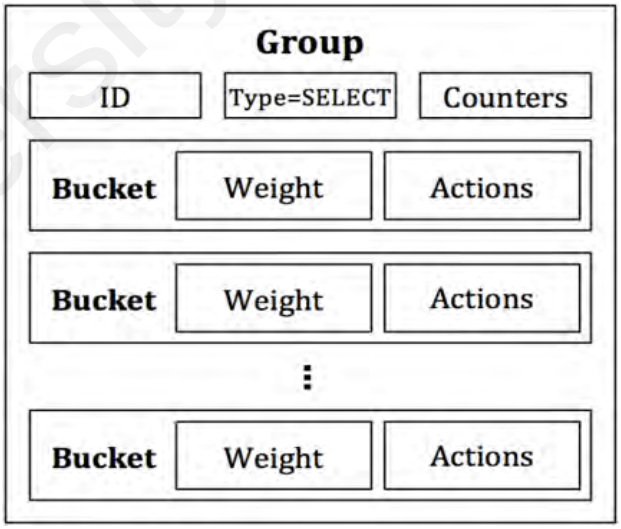


Figure 4.1: The SELECT Group (Izard, 2016)

Adopting the OpeFlow's *group table* to flow measurement, provides the feasibility to aggregate a set of flows into one group. Therefore, such an approach significantly saves number of requests and replies message, due to the measurement request and reply of the

group rather than each single flow. In this case, the optimal number of flows in every interval is captured. Figure 4.3 depicts the Wireshark file including request and reply captured packets. As can be seen in the figure, the total length of the request l_{rq} and reply messages l_{rp} is 122 bytes (the same as the PA in Chapter 3) and 218 bytes respectively.

Algorithm 1 mapping flows to group table

Input: $f = \{x \mid x \subseteq F\}$: set of flows, $G = (V, E)$: the network, $\delta = match$

```
1:  c = create(Group_i)              // creating group on the switch
2:  for each x ∈ f do
3:      if (isNew(x)) then           // check if the flow is new
4:          if (x[attribute] == δ) then   // check if the flow is matched
5:              Group_i ← x
6:          end if
7:      end if
8:  end for
9:  return c
```

Figure 4.2: Pseudo-code of Construct Group and Mapping Flows to the Group

Table 4.1 shows the reply message structure and size for a single flow. However, similar to PA, the reply message may split into multipart reply messages if the length exceeds 64Kb in the medium. The optimal solution for communication costs, message interaction and controller overhead (for each controller) in network *G* with an arbitrary set of flows *f* is formulated in equation 4-1, 4-2, 4-3, respectively, in out-of-band network deployment. The message interaction and controller overhead are the same in both in-band and out-of-band deployment. However, the total communication costs are highly dependent on the number of hops from the controller to the switch. Equation 4-4 shows the formulation for optimal communication overhead in in-band deployment.

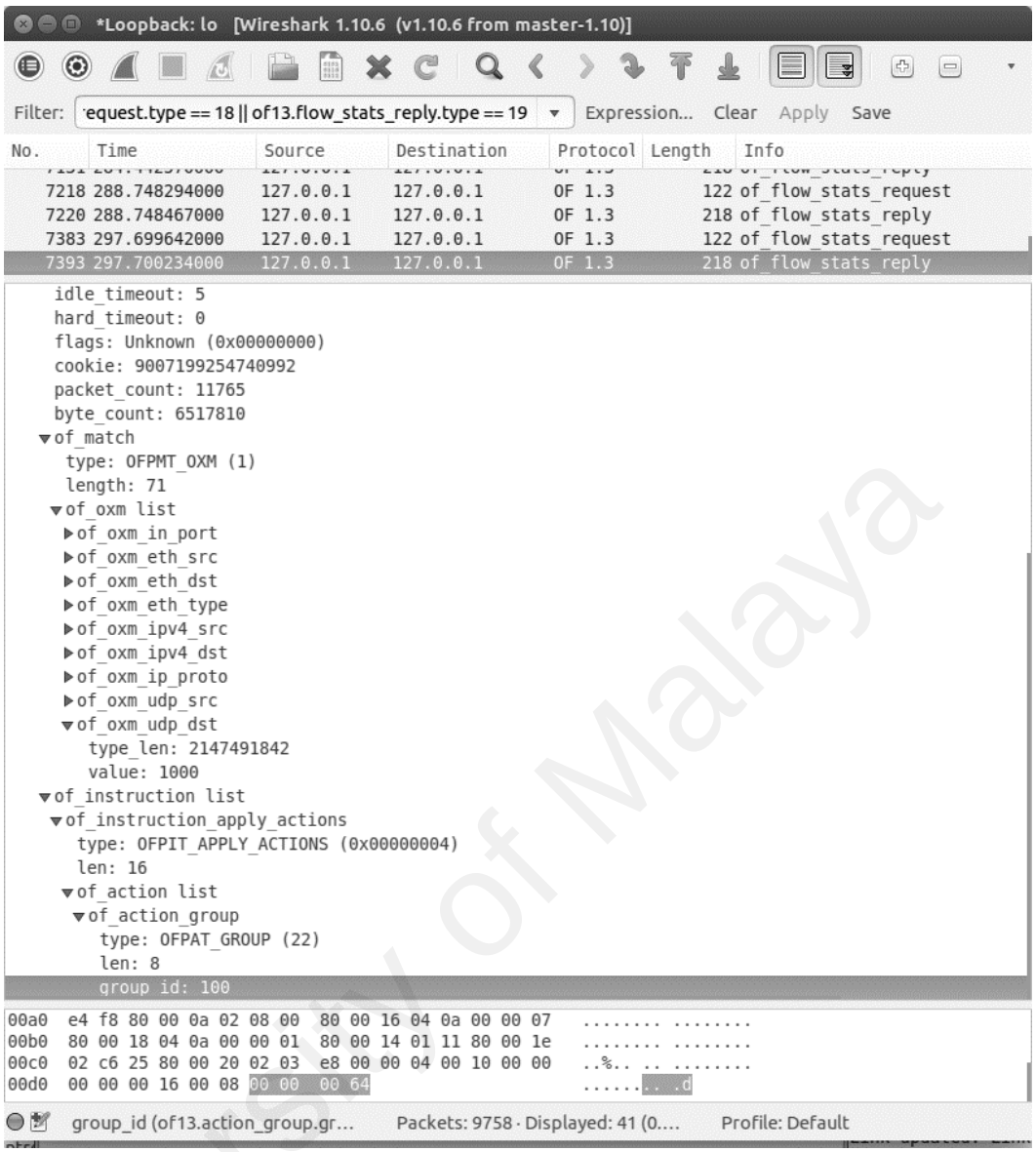


Figure 4.3: Wireshark file Including Request and Reply Captured Packets

Table 4.1: Request Message Structure and Length

Specification		Size (byte)
Header	Ethernet	14
	IP	20
	TCP	32
OpenFlow Payload	Multipart request	16
	Single Flow Statistic	136
Total		218

$$OptimalCost_{com}(\mathrm{f}) \cong l_{rq} + l_{rp} + \sum_{f_i} l_{sf_i} + (l_{udp} \times v_{kc}), \forall f_i \in f\ (0 < i < m), f \subseteq F, v_{kc} \in v, v \subseteq V \quad \textbf{4-1}$$

$$OptimalCost_{message}(c) = n_{rq} + n_{rp} + n_{udp}$$

Subject to:

$$\rho = \{m \mid m{:}\, N\}, p = \begin{cases} 1, & Cost_{com}(\mathrm{f}) \leq 65536bytes \\ m \times 2, & Cost_{com}(\mathrm{f}) > 65536bytes \end{cases} \quad \textbf{4-3}$$

$$OptimalCost_{controller}(f) = 3\theta + \sum_{f_i} \lambda_i, \forall f_i \in f\ (0 < i < m), f \subseteq F \quad \textbf{4-3}$$

$$OptimalCost_{com}(\mathrm{f}) \cong l_{rq} + l_{rp} + \sum_{f_i} l_{sf_i} + (l_{udp} \times v_{kc}) \times h_{\alpha\beta}, \forall f_i \in f\ (0 < i < m), f \subseteq F, v_{kc} \in v, v \subseteq V \quad \textbf{4-4}$$

4.1.1 Measurement Granularity

The granularity of the proposed solution is configured by setting the demanded specifications in the request message. Basically, the SDN controller is able speak to the switches by establishing a set of specifications in a request message. The switch then speaks back to the controller by sending the reply message as a response to the given specifications. This feature was adopted in the proposed solution by configuring a set of specifications such as IP src/des, port src/des, as well as many other match fields in different network layers. OpenFlow 1.3 present 40 fields to be used as match specifications. Therefore, the proposed solution sets all the active flows with a specific match (demanding granularities, for example UDP destination port 8080) to a group and the group is then pulled to measure its flows. As such, fine-grained flow measurement can be guaranteed when up to forth layer of OSI model (transport layer). However, the

more a request is fine-grained, the longer the request packet. The growth of the request packet depends on the length of the field match. The granularity of the measurement was set up to transport layer (UDP src/des). However, it is possible to simply modify it for more specifications. Table 4-2 shows the OpenFlow match fields along with their sizes in bits.

Table 4.2: OpenFlow Match Fields and length

Field	Length	Description
OXM_OF_IN_PORT	32	Ingress port. Numerical representation of incoming port, starting at 1. This may be a physical or switch-defined logical port.
OXM_OF_IN_PHY_PORT	32	Physical port. In ofp_packet_in messages, underlying physical port when packet received on a logical port.
OXM_OF_METADATA	64	Table metadata. Used to pass information between tables.
OXM_OF_ETH_DST	48	Ethernet destination MAC address.
OXM_OF_ETH_SRC	48	Ethernet source MAC address.
OXM_OF_ETH_TYPE	16	Ethernet type of the OpenFlow packet payload, after VLAN tags.
XM_OF_VLAN_VID	12+1	VLAN-ID from 802.1Q header. The CFI bit indicate the presence of a valid VLAN-ID, see below.
XM_OF_VLAN_PCP	3	VLAN-PCP from 802.1Q header.
OXM_OF_IP_DSCP	6	Diff Serv Code Point (DSCP). Part of the IPv4 ToS field or the IPv6 Traffic Class field.
OXM_OF_IP_ECN	2	ECN bits of the IP header. Part of the IPv4 ToS field or the IPv6 Traffic Class field.
OXM_OF_IP_PROTO	8	IPv4 or IPv6 protocol number.
OXM_OF_IPV4_SRC	32	IPv4 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV4_DST	32	IPv4 destination address. Can use subnet mask or arbitrary bitmask
XM_OF_TCP_SRC	16	UDP source port
XM_OF_TCP_DST	16	TCP destination port
XM_OF_UDP_SRC	16	UDP source port
XM_OF_UDP_DST	16	UDP destination port
OXM_OF_SCTP_SRC	16	SCTP source port
OXM_OF_SCTP_DST	16	SCTP destination port

Table 4.2, continued

OXM_OF_ICMPV4_TYPE	8	ICMP type
OXM_OF_ICMPV4_CODE	8	ICMP code
OXM_OF_ARP_OP	16	ARP opcode
OXM_OF_ARP_SPA	32	Source IPv4 address in the ARP payload. Can use subnet mask or arbitrary bitmask
OXM_OF_ARP_TPA	32	Target IPv4 address in the ARP payload. Can use subnet mask or arbitrary bitmask
OXM_OF_ARP_SHA	48	Source Ethernet address in the ARP payload.
OXM_OF_ARP_THA	48	Target Ethernet address in the ARP payload.
OXM_OF_IPV6_SRC	128	IPv6 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_DST	128	IPv6 destination address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_FLABEL	20	IPv6 ow label.
OXM_OF_ICMPV6_TYPE	8	ICMPv6 type
OXM_OF_ICMPV6_CODE	8	ICMPv6 code
OXM_OF_IPV6_ND_TARGET	128	The target address in an IPv6 Neighbor Discovery message.
OXM_OF_IPV6_ND_SLL	48	The source link-layer address option in an IPv6 Neighbor Discovery message.
OXM_OF_IPV6_ND_TIL	48	The target link-layer address option in an IPv6 Neighbor Discovery message.
OXM_OF_MPLS_LABEL	20	The LABEL in the first MPLS shim header.
OXM_OF_MPLS_TC	3	The TC in the first MPLS shim header.
OXM_OF_MPLS_BOS	1	The BoS bit in the first MPLS shim header.
OXM_OF_PBB_ISID	24	The I-SID in the firrst PBB service instance tag.
OXM_OF_TUNNEL_ID	64	Metadata associated with a logical port.
OXM_OF_IPV6_EXTHDR	9	IPv6 Extension Header pseudo-field.

4.2 Architecture of the framework

The architecture of the proposed framework consists of two stages: (a) a local controller design that describes the entire schema in layout-like steps, and (b) a core design that focuses on the design of the local controllers.

4.2.1 **Design of Layout**

Figure 4.4 depicts the schema of the proposed system layout. It consists of three layers: the OpenFlow network layer, the OpenFlow controller level, and a coordinator level on top of all the controllers connecting to the switches.

(a) The OpenFlow network level consists of all the low-level network entities such as hardware and software devices connected to the upper layer via the northbound interface.

(b) The controller level is the heart of the proposed design, where statistics of flows are collected by their associated local controller in any time interval, for example every second. Each controller that is associated to a flow is set by the top layer to request the flow statistics, and is responsible to collect, aggregate, and forward them to the upper layer. The controller associated to a flow is able to track a flow and instruct it. The upper layer mentioned above is the coordinator level, which has access to all controllers and is able to instruct the controllers.

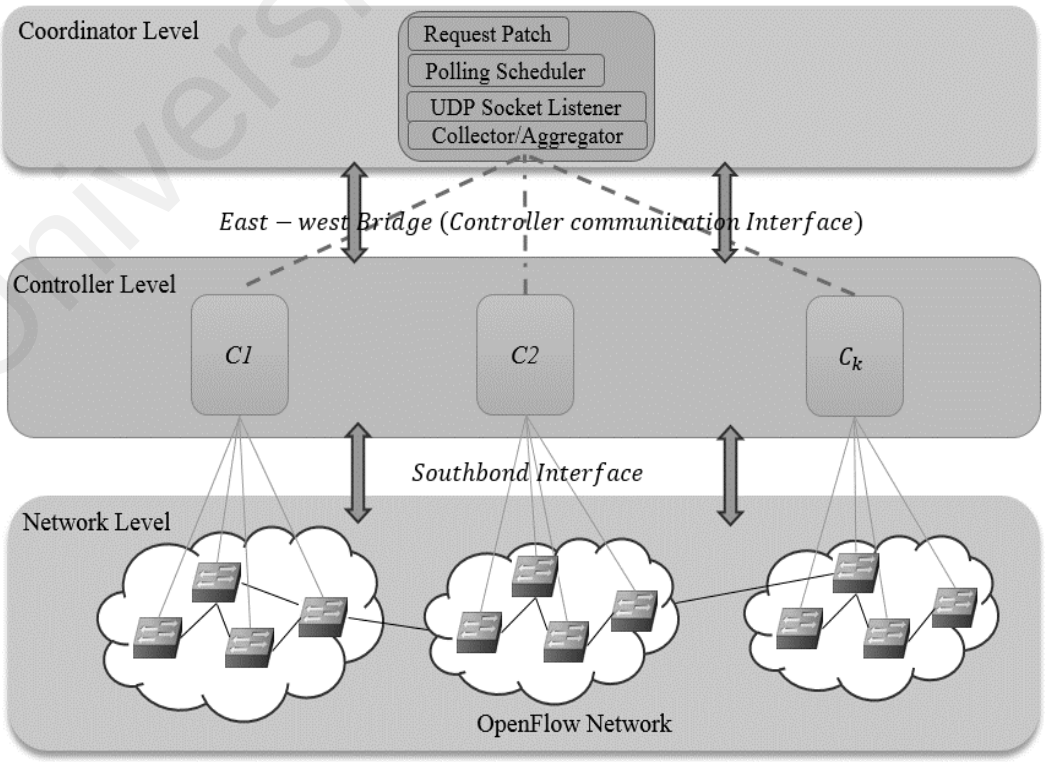


Figure 4.4: Schema of system layout

(c) The coordinator level is responsible to set controller(s) to request flow statistics through *Request Patch* in an arbitrary fashion such as fixed or adaptive polling, using *Polling scheduler*. The coordinator receives statistics from different controllers by *UDP Socket Listener* and accumulates them to shape a traffic matrix (TM) of demanded flows. This layer provides an east-west interface to interconnect the controllers and bridge all the gathered statistics to accumulate the demanded tasks. The east-west interface is often called east-west bridge where it is responsible for implementing the function of efficient communication, synchronisation and negotiation among multiple controllers (Xie et al., 2015).

4.2.2 **Local Controller Design**

Figure 4.5 shows the architecture of a local controller. There are four steps to accomplish the measurement task after a local controller receives a flow statistic request from the coordinator using the "Request Dispatcher" module.

(a) Flow Tracker: The first step is tracking all the flows (current/new flows in the controller domain) with a specific characteristic (user demands) that needs to be monitored.

(b) Group Maker: The second step is grouping all the flows that were specified earlier in the first step. This module utilises the "group table" feature in OpenFlow specification 1.3. It then instructs the switch(s) to modify the associated TCAM output group entry by sending a packet_out message to the switch(es).

(c) Query Maker: In the third step, switches are pulled with the exact match of the created group in the previous step.

(d) Collector: All the statistics in each time interval are aggregated by this module and sent to the top layer (coordinator). The process of sending aggregated stats is performed by a simple UDP datagram socket. Section 5.2.2.4 explains the time

delay and latencies caused by the synchronisation process (UDP socket), as well as its impact on accuracy of the measurement.

The proposed design is implemented as a northbound application on top of the controllers. The coordinator can access multiple local controllers by a simple application call. The proposed design can be implemented in various single and multiple-controller scenarios such as clustered, distributed, and hierarchical. Furthermore, it is able to accomplish almost all aspects of a monitoring system such as flow utilisation, measuring available bandwidth, packet-loss, link and packet delay.

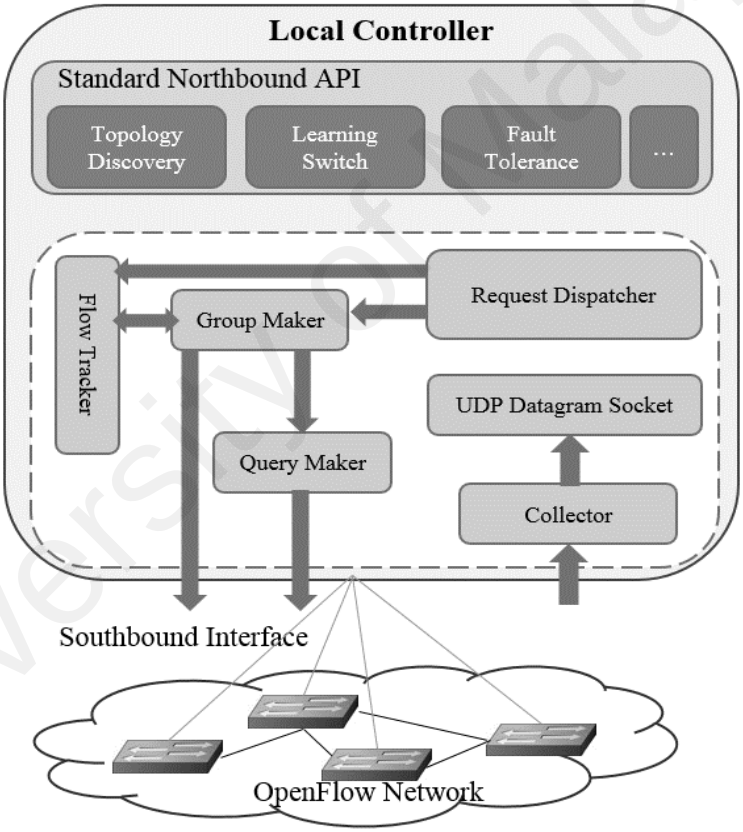


Figure 4.5: Local Controller

4.3 Cost-Effective Multi-Objective Controller (CEMoC)

Applying multiple controller may result in several unexpected performance degradations such as accuracy and overhead. Each switch can be attached to only one master controller, hereupon assignment of multi-controllers extremely effects on

overhead and accuracy. The assignment and re-assignment of controllers can be referred to as switch selection where the switch(s) may be attached to different controllers in different place. In addition, different deployment of such a scenario highly impacts several factors in the network such as node-to-controller latencies, network availability and performance metrics (Karakus & Durresi, 2017). Therefore, the controller (place of controller) fetching flow statistic plays a vital role in the accuracy of real-time monitoring as well as cost, especially in the in-band deployment. In addition to controller assignment, different placement of the coordinator can cause extra cost as well as receiving unsynchronized stats which lead to inaccuracy of results.

As it was mentioned earlier, the monitoring and routing traffic shares bandwidth along the same path. Therefore, this deployment requires careful planning and precise placement of controllers in the network. Thus, the number of network elements such as switches, routers, and cables can highly effect on the communication costs as well as the accuracy of result for real-time measurement purpose. Although, certain networking factors such as propagation delay can negatively mutate statistical accuracy. Let donate $h_{\alpha\beta}$ as the number of hops from switch v_α to controller c_β and $h_{\beta k}$ as the number of hops from the controller c to the coordinator k. let w is the cost of communication for polling flows from a switch. let donate d as the cost for each controller to be assigned on the switch β and q is the cost of communication from the controllers to the coordinator respectively. Given propagation delay $\mu_{\alpha\beta}$ for each source-destination pair $v(\alpha, \beta)$, equation (4-5 to 4-8) describe the integer linear programing of the problem formulation in in-band deployment.

The objective function given by equation 4-5 describes the problem formulation of costs, which means selecting the most appropriate switch(s) to be polled (the switch that covers most flows) by w, and the best controller(s) to be assigned given by d_{ck} on the switch(s)

for polling in terms of minimum communication, propagation delay and controller overhead (costs). It also presents the communication between the coordinator and all the controllers. Equation 4-6 refers to the constraint for selecting a switch in which at least one switch is selected. Equation 4-7 explains the constraint for the controller to be assigned to only one switch. Equation 4-8 forces selecting all the source-destination pairs $v(\alpha, \beta)$ with the least propagation delay. The binary variable used in the formulation is explained as follows:

- A binary variable x_β represents whether to poll flow from switch β or not, 1 if it is polled
- A binary decision variable y_{ck_β} represents whether a controller ck is assigned on the node β nor not, 1 if it is assigned.

$$w_i = l_{rq} + l_{rp} + \sum_{f_i} l_{sf\,i}\,, f_i \in f\,, f \subseteq F\ (0 < i < m)$$

$$d_{ck} = min \sum_{ck \in C} \sum_{\beta \in V} w_{i_\beta}\,, \forall ck \in C, \forall \beta\ \in V$$

$$co_{c_{ck}} = \min \sum_{ck \in C} (3\theta + \sum_{f_i} \lambda_i)\ \,, \forall f_i \in f\ (0 < i < m), f \subseteq F, \forall ck \in C$$

$$q = \min \sum_{ck \in c} \sum_{h_{\beta k}} l_{udp}\,,\ \forall \alpha, \beta\ \in V$$

$$\min \sum_{\beta \in V} \sum_{i \in f} w_{\beta_i} x_{\beta_i} + \sum_{\beta \in V} d_{ck_\beta} y_{ck_\beta} + \sum_{\alpha \in |V|} q_\alpha\,, x_{\beta_i}, y_\alpha, z_\alpha \in \{0,1\}, \forall ck \in C, \forall \alpha, \beta \in V \quad \textbf{4-5}$$

Subject to:

$$\sum_{\beta \in |V|} \sum_{i \in f} w_{\beta_i} x_\beta\ \geq 1, \forall \beta \in V\,, q\ \in \{0,1\} \quad \textbf{4-6}$$

$$\sum_{\beta \in |V|} y_{ck_\beta} \leq 1, \forall \beta \in V, y \in \{0,1\} \quad \textbf{4-7}$$

$$\sum_{h_{\alpha\beta} \in |V|} \mu_{\alpha\beta} \leq \min\{v(\alpha,\beta)\}, \forall \alpha, \beta \in V \quad \textbf{4-8}$$

The minimisation problem defined above is a weighted set-cover problem that is proven to be NP-hard and requires heuristics to approximate the performance. It should be noted that propagation delay is computed for a link between two switches, as it is determined by LLDP send/receive times minus the delay between both switches and the controller. However, this delay does not cause a significant effect in datacentres as it is in the order of microseconds.

The easiest way to solve the problem of weighted set-cover is to apply a brute force search algorithm known as exhaustive search, which enumerates all possible candidates for the solution and checks its correctness. Although the brute search algorithm is simple and always finds the solution, its time complexity is $O(2^{m+n})$, which is exponential to the number of flows and switches $m\ and\ n$ respectively that is neither scalable for datacentres nor for ISPs. Hence, the approximation technique is required for large-scale networks.

To solve the problem above an eager-greedy algorithm was applied, which is an approximation technique adopted from (Lim et al., 2014). This algorithm implements a priority queue to alleviate time complexity to $O(m \log n)$. The algorithm selects the most cost-effective switch(es) that covers all demanding flows based on their given weight (w_i). Figure 4.6 (Algorithm 2) shows the steps involved in the selection of switch(es). In each iteration, it calculates the minimum associated weight (shortest path) for all demanding flows in step 4. It then identifies the sets with the largest number of uncovered items in step 6 and puts it in the output as a group in step 7. If the algorithm finds more

than one sub-covered set (step 5), it selects the subset with the least propagation delay in step 6. Figure 7 shows the entire architecture of the CEMoC design.

This thesis also proposes an algorithm that assigns a controller to a switch for polling purposes. It should be noted that only one controller can be assigned to a switch with the master mode.

Algorithm 2 The Eager-greedy approach

Input: f: sets of flows, $f\ (0 < i < m), f \subseteq F$; w: weight of polling set f.
Output: A: set of groups for polling $A\ = \{x \mid x \subseteq F\}$

1: $A \leftarrow \{\}$
2: Covered $\leftarrow \{\}$
3: **while** covered $\neq f$
4: $j \leftarrow$ calculate (min w_{if})
5: if ($|max_{j\epsilon[0..m]}\{f_j - F\}| > 1$)
6: $c \leftarrow \min \mu_{\alpha\beta}(max_{j\epsilon[0..m]}\{f_j - F\})$
7: $A \leftarrow A \cup c$
8: covered $\leftarrow$ covered $\cup\ f_j$
9: **end while**
10: **return** A

Figure 4.6: Pseudo-code of Eager-greedy approach

Figure 4.7 (Algorithm 3) illustrates the steps involved in the selection of the controller. The main loop iterates for $O(n)$ time where $n = |A|$ which is the number of polling set(s). It then calculates the nearest controller with the least CPU load to the polling set (switch(s)) in step 3 with the time complexity of $O(m^2)$ where m=|C|, which is the number of controllers. In case of finding more than 1 controller for a set, it selects the one nearest to the coordinator's location. Figure 4.8 illustrates the entire flow process of the CEMoC. As shown Figure 4.8, algorithm 2 and 3 run only once upon a measurement request to the coordinator. Then the coordinator signals the selected controllers and polling set(switches) to be polled. The life line of controller(s) continues for the number of iterations. Therefore, every controller sends stat requests and receives stat replies until the user signals the termination of the process.

Algorithm 3 Controller selection

Input: c: sets of controllers, $c\ (0 < i < n), c \subseteq C$; A: set of groups for polling $A\ = \{x \mid x \subseteq F\}, \forall \alpha, \beta \in V$

Output: B: sets of controllers, $c\ (0 < i < n), c \subseteq C$

B, n ← {}
foreach polling set in A
    j ← calculate (min co_{cA} *and* min $hA_{\alpha\beta}$)
    if ($|j_A| > 1$)
        n ← calculate (min $j_{A\beta k}$)
    B ← $B \cup n_j$
    $A \leftarrow A \cup c$
end foreach
return A

Figure 4.7:Pseudo-code of Controller Selection

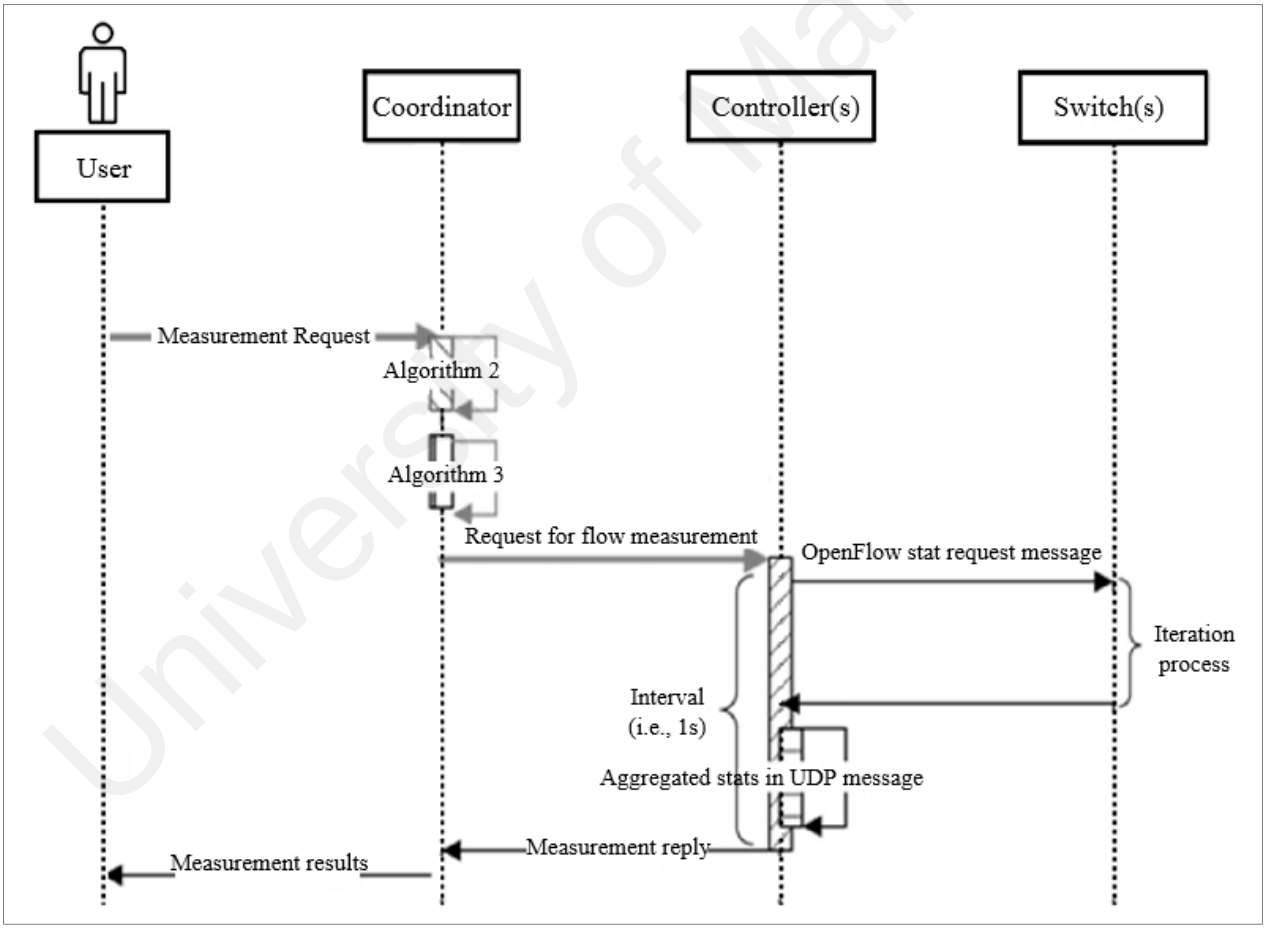


Figure 4.8: Entire Flow Process of the CEMoC

4.4 Summary

This chapter proposed a cost-effective flow measurement framework aiming to effectively minimise various costs such as communication, message interaction and

controller overheads that are associated with the measurement of flows in single and multiple-controller scenarios in different network deployment models (out-of-band and in-band). The chapter first elaborated on the proposed solution that aims to minimise the costs of pulling flow statistics using grouping flows and pulling groups as the target rather than flows. It described how the proposed solution can reduce the length of reply messages using *Select* group requests, and further formulated the costs when applying the proposed solution. Moreover, the architecture of the proposed framework is described in two designs, namely the layout and the local controller design. The chapter explained that requests are sent from the coordinator to the switch(es) and then reply messages are issued from the switch(es) and sent to the controller. Finally, it proposed a multiple-controller design and a controller selection algorithm that optimise the selection of the switches for polling and the location of the coordinator.

The next chapter evaluates and validates the proposed framework through various experiments to show its capability and effectiveness to fulfil the targeted objective.

CHAPTER 5: PERFORMANCE EVALUATION

One of the most important processes in conducting research is to evaluate the proposed design, in order to validate its outcome and highlight its feasibility and suitability. This chapter explains the systematic evaluation phases to accredit the evaluation and analysis of the performance. The chapter thoroughly elaborates on the different stages of performance evaluation to perform a realistic assessment of the proposed framework, and subsequently discusses the findings. The remaining chapter is outlined as follows:

Section 5.1 explains the evaluation setup by introducing its environment, datasets, topologies and tools. Moreover, it presents the benchmarking methods, their objectives, and the reason for selecting these benchmarks. Section 5.2 introduces different experiments trough emulation and simulation. It then describes the findings from the experiments and compares them to the benchmarks. This is followed by a comprehensive analysis and discussion of the findings from the different experiments. Section 5.3 aims at the verification of our simulator with which the second experiment was performed. Section 5.4 offers a comprehensive discussion of all findings from the different experiments, including their pros and cons. Finally, section 5.5 provides concluding remarks for the whole chapter.

5.1 Evaluation Setup

This part comprehensively describes the experiment setup, the datasets used in the evaluations, the set of tools to exploit the findings or capture the data/results, and the data collection models.

This study presents three evaluations through extensive experiments. To ensure the validity and reliability of the findings, the experiments were iterated 10 times. Findings from the first five iterations demonstrated a linear trend associated with an independent

variable such as *number of flows*. Thereby, to ensure the linearity of trends, all the experiments were repeated another five times, however, the findings demonstrated almost identical values with a negligible variation below 0.05 percent. The results and findings from the experiments are presented by averaging all iterations in every experiment. Some experiments share similar requirements or datasets while others require different specifications and datasets.

5.1.1 Experimental setup

In order to evaluate the performance of the proposed model, three evaluations were conducted through emulation and simulation; the first two experiments employ emulation, and the last experiment is conducted via simulation. The environmental setup varies based on the experiments. For instance, experiment 1 uses a lab setup environment, while the second experiment is evaluated in Amazon EC2 due to the heavy workload and traffic dataset. The emulation of a real network is resource hungry and requires a huge memory and CPU capacity (Su et al., 2015), therefore the emulation was carried out with limited traffic in a Mininet emulator. Similar to previous works (Su et al., 2015; H. Xu et al., 2017), a large-scale experiment (experiment 3) was performed through simulation as the experiments only cares about the active flows, shape of the topology, the links, and their forwarding paths. Therefore, the large-scale experiment was carried out using a trace-driven simulator built only to simulate the event of flow arrival and expiration in the network. However, the accuracy-related experiment was only conducted by emulation, as simulation is incapable of emulate the real behaviour of a network such as delays or varied network latencies.

5.1.2 Experiment Tools

a) The SDN controller in this study is Floodlight (BigSwitchNetworks, 2016). This thesis employs Floodlight controller because of its unique features and several

advantages over other SDN controllers. The following explains the reasons for choosing floodlight:

i. Floodlight is an open source SDN controller that supports OpenFlow version 1.3. The controller can also be extended to support version 1.4 and 1.5 of OpenFlow through the experimenter.

ii. Floodlight has been developed as a modular system that is highly flexible and can be simply extended and enhanced.

iii. Floodlight supports northbound Full-REST API, which can be accessed through HTTP request.

iv. Application are implemented as northbound API and placed on top of a logically centralised controller. Consequently, other network applications can use the API by a simple http request anytime and anywhere.

v. Floodlight supports full state synchronisation, which allows several instances of it to be connected in a cluster or distributed manner (multiple-controller).

b) The prototype of the proposed design was implemented as a northbound API module of a floodlight controller.

c) This thesis employs Mininet version 2.2.1 for emulating the entire network and its characteristics such as links, topology, latencies and so on. Mininet was chosen because it is the most common emulator and is used in most research. Besides, Mininet is the most compatible network emulator to support OpenFlow version 1.3. Furthermore, it uses Linux containers to create a virtual network to emulate hosts, which allows the entire network to be emulated in a single computer. Therefore, the emulated network is isolated from other performance sensitive applications in the memory. Moreover, Mininet uses Open vSwitch (OVS) as a virtual switch to emulate the behaviour of the real switch. Open vSwitch version 2.5.2 (OpenvSwitch, 2017) which is the latest version at the time of conducting the experiments.

d) D-ITG (Botta et al., 2012) network traffic generator was employed to generate realistic network traffic with various patterns. D-ITG has been proven to perform in a more reliable and scalable way than other traffic generators (Megyesi et al., 2017).
e) Wireshark was used to capture the real traffic volume of the original datasets and the traffic generated by the proposed model and other benchmarking methods.

5.1.3 **Datasets**

One of the most important stages in the evaluation process is to validate the proposed model under a concrete and realistic workload. This evaluation applies three different types of traffic patterns as datasets. The rationale behind employing various datasets in the validation process is to understand the behaviour of the proposed framework and the benchmarking methods. It also shows how the models react with different traffic workloads and what types of model is suitable for datasets. The following presents the workloads employed in this research.

a) Constant Bit Rate (CBR)

In this model (Megyesi et al., 2016), all the hosts generate traffic flows in a uniformly random manner which is constantly increased in rate and size. All hosts generate new flows every second and send them over all other hosts. CBR is a reliable candidate to validate the costs because it constantly increases the distribution of flows in the network. Therefore, it simply discloses the variation of overhead changes generated by requesting flow statistics. Moreover, it demonstrates the relationship between flow numbers and overhead ratio among different methods. The first two experiments apply this dataset with different flow distribution ratios. The dataset is further explained in each experiment with more specifications.

b) Variable Bit Rate (VBR):

It follows the traffic pattern introduced in (Megyesi et al., 2017) (shape and scale) for all generated flows in the network. VBR applies D-ITG pareto distribution for the packet 's inter-departure time. Similar to (Megyesi et al., 2017), this evaluation uses $\varphi = 1.75$ as the shape parameter for all flows, and a random scale parameter Υ from the range 0.5ms to 1ms. This type of traffic pattern is suitable to stress the accuracy of flow statistics as it fluctuates highly and leaves numerous traffic spikes in the network. Therefore, the evaluation employs this workload to demonstrates the performance on the proposed model on accuracy and error ratio. This study employs VBR in experiment 2 for evaluation of the accuracy.

c) Real dataset workload:

A publicly available data packet trace collected from a university datacentre in (Benson et al., 2010). The evaluation applies a real dataset from a datacentre in the university of Wisconsin to realise the performance and suitability of the proposed framework under a real workload. The workload is employed for the large-scale evaluation in experiment 3.

5.1.4 Topology

The evaluation of this study deploys different topologies to demonstrate that the proposed framework is a scenario-independent model, and thus can be applied to various conventional datacentre topologies. Furthermore, using various topologies, the evaluation validates the concreteness of findings and the proof of concept of the framework. The topologies are specified in more detail in every experiment. Below is a general description of the topologies applied in this research work:

a) Synthetic topology:

This topology implements a single pod of the fat-tree topology model. The topology includes four switches that consist of two edges, and two aggregation switches of a 4-pod fat-tree, which is a common topology used in datacentres. The topology is used in the first experiment where the proposed model is evaluated on the single controller scenario with out-of-band network deployment. As out-of-band is the simplest network deployment model (i.e. switches are directly attached to the controller using the network control links), this topology aims to demonstrate the very pure behaviour of the proposed framework and benchmarks regardless of the sophistication of the network configuration and topology.

b) k-pod fat-tree:

This is a common topology deployed in many datacentres and is organised in a tree-like structure. The k-pod fat-tree topology has the properties that make it suitable for data centre networks (Bari et al., 2013). Figure 1 in Appendix B depicts the schema of a k-pod fat-tree topology, k=4.

c) 2-tier fat-tree:

This study adopts this topology from a publicly available dataset that can be found in (Benson et al., 2010). The topology is a datacentre network located in the university of Wisconsin, USA. It employs a similar topology to a conventional 2-tier fat-tree type, which only uses core and edge switch in the network deployment. Interested readers are referred to (Bari et al., 2013). However, unlike the canonical 2-tier topology architecture that employs Top-of-Rack switches (each switch connects to a rack of 20-80 serves), this topology uses Middle-of-Rack switches that connect a row of 5 to 6 racks with the potential to connect from 120 to 180 servers. Figure 2 in Appendix B shows the schema of the 2-tier datacentre topology.

5.1.5 Performance Metrics (Parameters)

The performance of the proposed framework is evaluated with standard parameters of different overheads and accuracy. The term overheads herein refers to various overhead such as communication, message interaction and controller.

In order to highlight the effectiveness of the proposed framework on different overheads and accuracy, the experiments aim to satisfy the following requirements:

i. To explain the communication overhead and its relation to demanded flows and total number of flows in the network and compare them to the benchmark methods.
ii. To highlight the message interaction overhead and the impact of different numbers of flows on this cost and compare them to benchmark methods.
iii. To elaborate on the controller overhead in the proposed model and compare that to benchmark methods with different numbers of flows in the network.
iv. To investigate the relationship between different overheads and the impact of flow increment or decrement in the links and switches.
v. To observe the accuracy (error ratio) of flow measurement in comparison to the actual flow utilisation by the proposed framework and investigate the impact of controller number on the accuracy.

5.1.6 Comparison to the current State-of-the-art: Benchmarking Methods

The performance evaluation and verification of the proposed method is carried out by current state-of-the-art pull-based per-flow measurement/monitoring methods described comprehensively in Chapter 2. The following elaborates on the benchmarking methods and the reason for choosing these benchmarks:

i. Single flow Stat-Request (SSR): The method is an OpenFlow native approach for obtaining flow statistic requests. Many pull-based studies have exploited SSR to measure network flows. Likewise, the evaluation also employs SSR because it directly discloses the relation between the required flow statistics and the different overheads. The method is also expected to generate high messaging and communication overheads.

ii. Polling All-flow Stat-Request (PA): The approach polls all the current flow stats of a switch with one request and reply message and has been widely implemented in the literature. The aim of applying this method in the evaluation is to manifest the effectiveness of aggregating all flows within a single request. Also, it reveals the impact of flow distribution in the network on different overheads. The expected behaviour of this method is high overheads for the communication and controller that is seen as the side effects. However, it significantly decreases the message interaction overhead.

iii. CeMon: The model is the most similar to the proposed framework in this study as it employs a combination of SSR and PA models. It is expected to have both the advantages and disadvantages of SSR and PA.

5.2 Result and Discussion

The section presents the results and analysis of the experiments (i.e. emulation and simulation) of the proposed framework. The main aim of this section is to expose the outstanding performance of the proposed model compared to the benchmarking methods. The section presents the findings and discussion for each experiment, within the corresponding performance metric of the experiment. The section also demonstrates different experiments with distinct objectives. Furthermore, it elaborates on a simulation experiment in which the dataset employed is a real datacentre workload. To validate the simulation results, the chapter is followed by a detailed performance analysis through

statistical modelling. At the end of the section, the evaluation result and findings are discussed. It should be noted that the emulation experiments on CBR traffic were iterated 10 times; as such, no change was observed from the findings of different iteration in the first experiment. Also, no tangible changes were observed in the overheads. The variance of findings of different iterations reported, is below 0.05. However, findings from accuracy (error ration) in the second experiment are obtained through only one run/iteration as different iterations on VBR traffic cause different results. This is because, the network latency in each time of running the emulation is not a constant value. Also, the ration of traffic shape is a random scale parameter Υ from the range 0.5ms to 1ms, as mentioned in the section 5.2.3. Findings of the proposed method from this study are labelled and shown as "Proposed method" in the experiment 1, whereas experiments 2 and 3, label and depict the findings from the proposed method as "CeMOC". This is because experiments 2 and 3, evaluate the decentralized (multiple) controller scenario.

5.2.1 **Experiment I: Single controller with out-of-band deployment**

This section aims to discuss the experimental evaluation of the proposed model in out-of-band network deployment with a single controller. The experiment comparatively investigates the relationship between overhead factors and flow number, by means of applying the proposed model to the synthetic topology and comparing the results with the benchmark methods. The objective of this experiment is to measure the utilisation of all flows passing through the link *pi* shown in Figure 5.1.

The experiment was conducted in a lab environment on a server with Intel(R) Xeon(R) E3-1270 processor 3.50Ghz and 16GB RAM. Figure 5.1 shows the synthetic topology used in this experiment. The experiment employs the CBR dataset as workload, with the maximum universe flow F=2000, the initial number of arbitrary flow passing through path p_i is f = 10, and the initial flows' number in the node *v* is 100 with the increase ration

of 66%,40%,28%, and 20% in the next 4 iterations. CBR is a valid traffic pattern that demonstrates a constant increase ratio by which it effectively discloses the incurred variation associated with the increment ratio. Controller *C1* polls the switch that is directly attached to it. To observe the effectiveness of the proposed method in a single controller, the obtained results are compared to two native OpenFlow polling methods namely SSR and PA and a similar work CeMon. The flows are generated in a uniformly random fashion using D-ITG. Table 5.1 shows the summary of the specifications in this experiment.

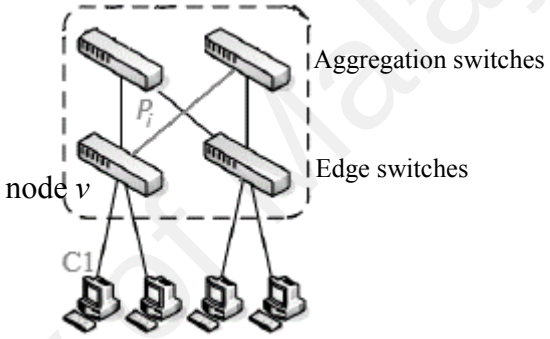


Figure 5.1: Synthetic Topology: Composed 1 pod consists of 2 edges and 2 aggregation switches with one controller.

Table 5.1: Experiment specification details

Specification	Details
SDN emulator	Mininet v. 2.1
Switch type	Open vSwitch(OVS) v. 2.5.2
Traffic generator	D-ITG
Traffic Type	Randomized TCP/UDP
Synthetic network topology	1 Pod of k-pod fat-tree k = 4
Number of flows in *pi*	10, 20, 30, 40, 50
Number of flows in the switch	100, 200, 300, 400, 500

This evaluation concentrates on out-out-band network deployment with a single controller. The results of this evaluation are shown in the average of 10 times iteration. All findings from this experiment are analysed and discussed in this section.

5.2.1.1 **Communication overhead**

Figure 5.2 shows the communication overhead in a single controller scenario with out-of-band deployment.

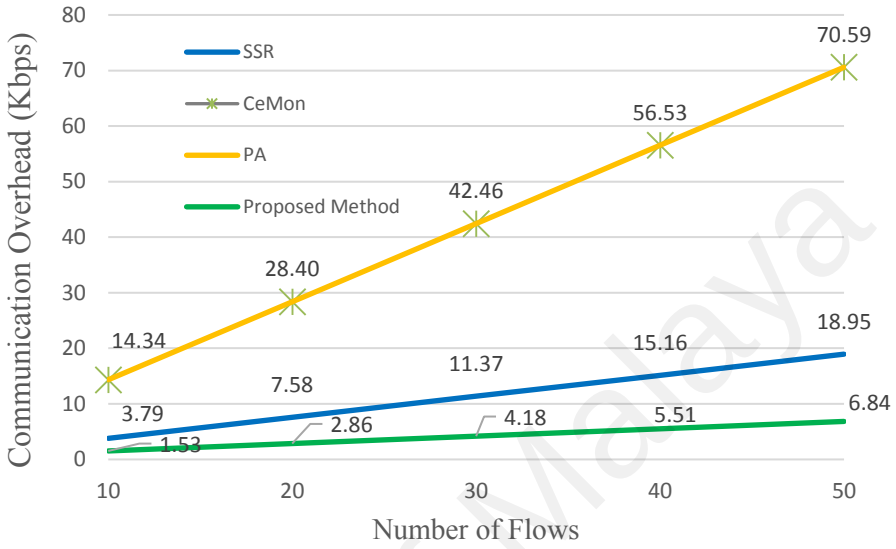


Figure 5.2: Communication overhead in single controller scenario with out-of-band deployment

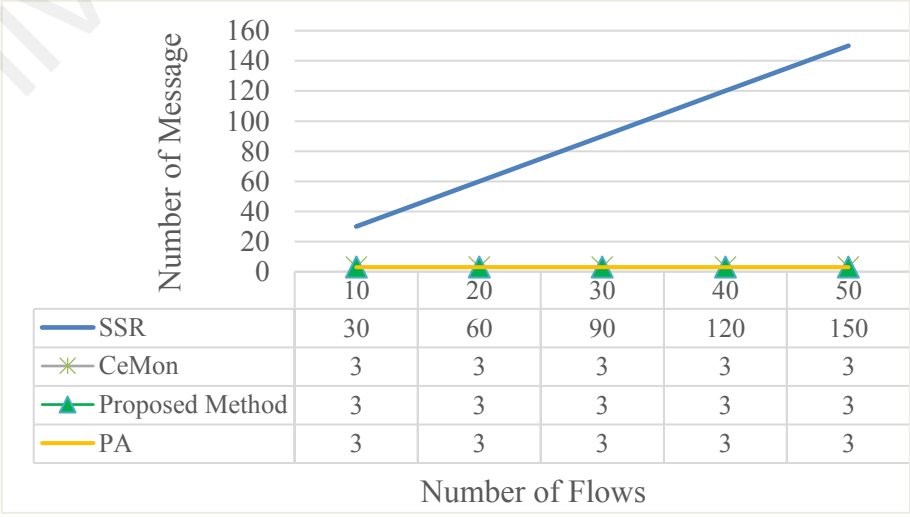
In general, the results obtained from this experiment indicate a notable reduction over all the compared methods. In particular, the proposed solution reported 82% reduction against SSR. It also saves up to 161% compared to both PA and CeMon methods. Basically, applying the 'polling all' approach can reduce the communication overhead that is strongly associated with the total number of flows in the switch, which is ten times bigger than the actual demanded flows. As can be seen, the sharp rise in PA and CeMon has a direct correlation to the total number of flows in the switch in every interval. This behaviour occurs in all methods that fully or partially apply the *polling all* strategy in their design. CeMon implements a combination of the *polling all* and *single flow request* approaches, in which it pulls all flow statistics in the previous intervals (already covered flows) and utilises single requests for new flows in the current interval. However, this strategy is highly scenario-dependent, which is either suitable for those networks with small rates of new flow arrival or multiple switch selection for polling. According to the

observed finding, it can be concluded that the reduction ratio in this metric (reducing the overhead) will be better in a network with a higher number of flows.

5.2.1.2 **Message Interaction Overhead**

Figure 5.3 shows the message interaction overhead in single controller scenario with out-of-band deployment.

The results obtained from the message interaction overhead demonstrate that the proposed method and CeMon archive the optimal result, which is found in the PA approach for message interaction costs. The costs for PA, CeMon and the proposed method remain constant and proportionally increase with the number of controllers. As there is only one controller in this case study, the proposed method generates three messages (request, reply and synchronised message) for every interval PA. CeMon also achieves the optimal number of message interactions as a result of applying "polling all approach". However, similar to communication costs, the result is highly dependent on the scenario and environmental factors such as the number of selected switches to poll or the arrival of new flows. SSR reported the worst approach in this overhead as it generates three messages for each flow.



	10	20	30	40	50
SSR	30	60	90	120	150
CeMon	3	3	3	3	3
Proposed Method	3	3	3	3	3
PA	3	3	3	3	3

Figure 5.3: Message Interaction in single controller scenario with out-of-band deployment.

5.2.1.3 **Controller Overhead**

Figure 5.4 shows the controller overhead in single controller scenario with out-of-band deployment.

As observed in the graph, the proposed method achieves the least controller overhead as it only processes statistics associated with the demand flows (those flows that contribute to the utilisation of p_i). CeMon and PA reported the highest overhead with an explicit increase of the number of flows. This is because these two methods send all the current flows in the switch to be processed in the controller. SSR are distinctively superior to CeMon and PA as a result of sending less flows to be processed. This reduction in flow numbers is due to polling less flow statistics. Thus, the fewer flow statistics to poll, the less overhead to be generated.

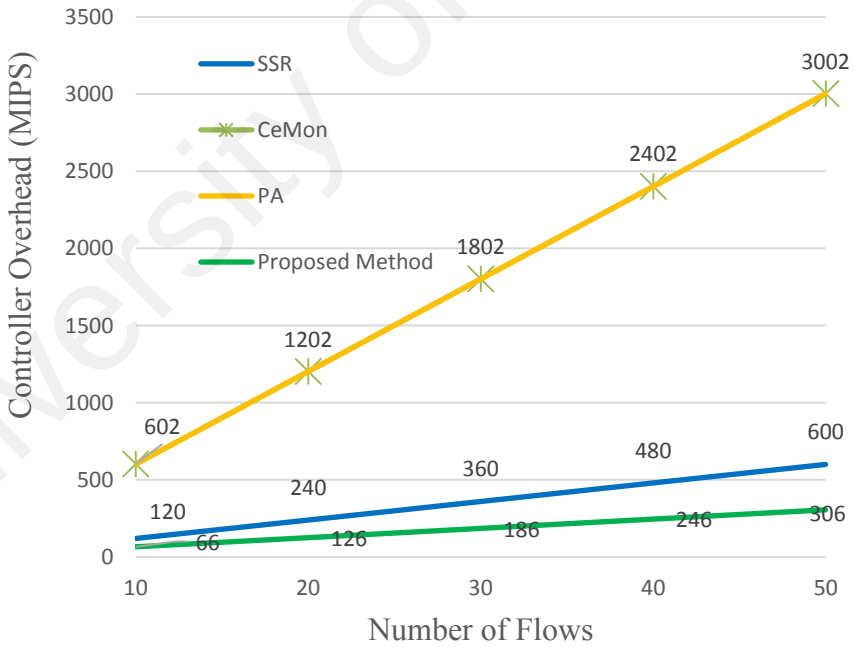


Figure 5.4: Controller Overhead in single controller scenario with out-of-band deployment.

However, it is observed that the proposed method significantly saves the overhead by 63% over SSR as a result of reporting the precise flow statistic as demanded. From the observation, it can be inferred that the number of message interactions and flows has an

explicit relationship to the controller overhead, and that the controller overhead increases according to the increasing message interaction overheads and number of flows. Therefore, similar to the communication overhead, the controller overhead is expected to save more in comparison to other methods when the number of flows in the network increases.

5.2.2 Experiment II: Multiple-controller (distributed controller) with in-band deployment

This section aims to discuss the experimental evaluation of the proposed model in in-band network deployment with a decentralised (multiple) SDN controller. It explores the effects of multiple SDN controller on various overhead factors and accuracy. The objective of the experiment is to measure all UDP flow traffic with a specific destination port number (i.e. 3660) in the datacentre network.

Table 5.2: Specification of the experiment

Spec	Type
Host vCPU	16
Host Memory	64Gb
Host OS	Ubuntu 16.04 Server
Mininet version	2.2.0
OVS version	2.5.2
Floodlight version	1.2
Traffic Generator	D-ITG version 2.8.1
Traffic Type	CBR

Network Topology	Number of Host	Switches			Number of Flow
		Edge	Aggregation	Core	
Fat-tree k-pod (k=4)	16	8	8	4	240, 480, 720, 960, 1200, 1440

As the scale of the experiment is large and requires high computation and memory capacity, it was conducted in an Amazon EC2 m4.4.xlarge instance with the Ubuntu 16.04 Server. The topology employed is k-4 fat-tree (shown in Figure 2, Appendix B) with CBR

workload. Additionally, a VBR dataset was employed to carry out the examination of accuracy performance metrics. All hosts generate VBR traffic to others. The detailed specification of the experiment is shown in Table 5.2. The findings of the proposed method for multiple SDN controller are labelled in CEMoC as presented in section 4.2.

5.2.2.1 **Communication Overhead**

Figures 5.5-5.8 show the communication overhead with different numbers of controllers. Obviously, the CEMoC is constantly superior to all other methods with different numbers of flows. A linear increase is observed in all methods as the numbers of flows grow. However. CEMoC reported the lowest communication overhead by maximum 410Kbps when the flow number is 1440. The finding shows that CEMoC saves up to 97% and 138% in comparison to PA and SSR, respectively. However, the findings demonstrate that the increment rate of communication overheads varies with different numbers of controllers. As can be seen in Figures 5.5, 5.6, 5.7 and 5.8, SSR, PA, and CEMoC stand out as the worst and the best methods in a descending order. However, CeMon reports different behaviour as it gradually decreased communication overhead when the number of controller decreases. Figure 5.9 explains the average growth ratio of communication overhead for each method over CEMoC; SSR and PA show a constant growth rate, whereas CeMon demonstrates the least increment rate with 51% in single controller mode, and the biggest change when there are four controllers in the network. This is because CeMon selects both core and edge switches to use polling all approach and applying single flow request respectively.

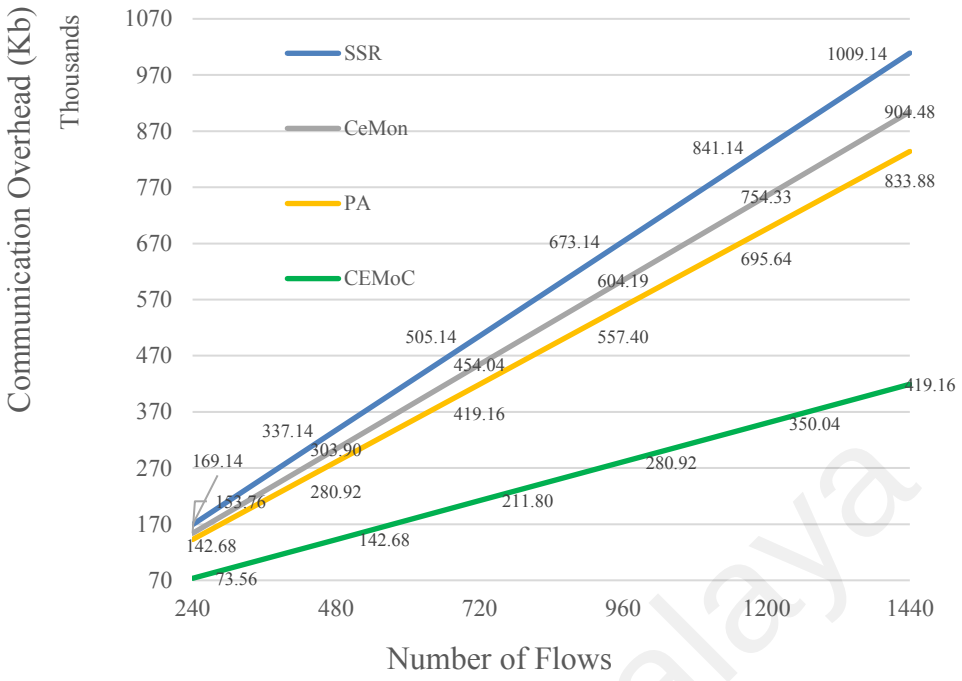


Figure 5.5: Total Communication Overhead with four Controllers

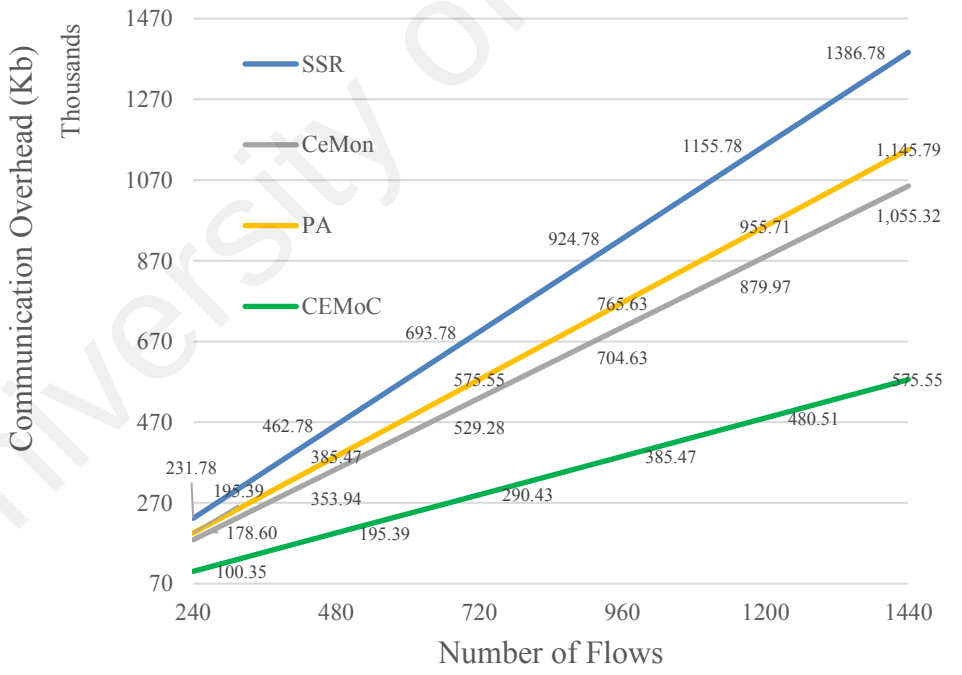


Figure 5.6: Total Communication Overhead with three Controllers

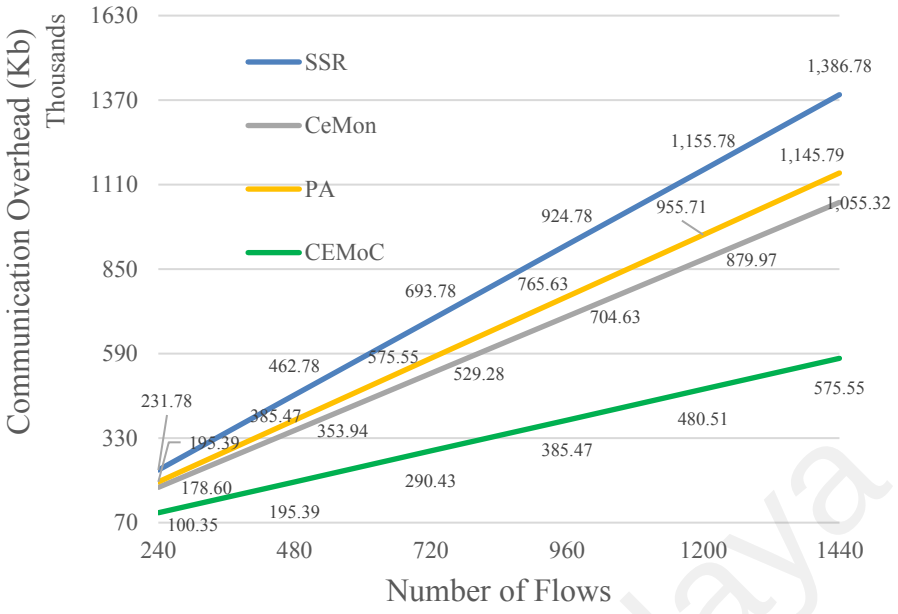


Figure 5.7: Total Communication Overhead with two Controllers

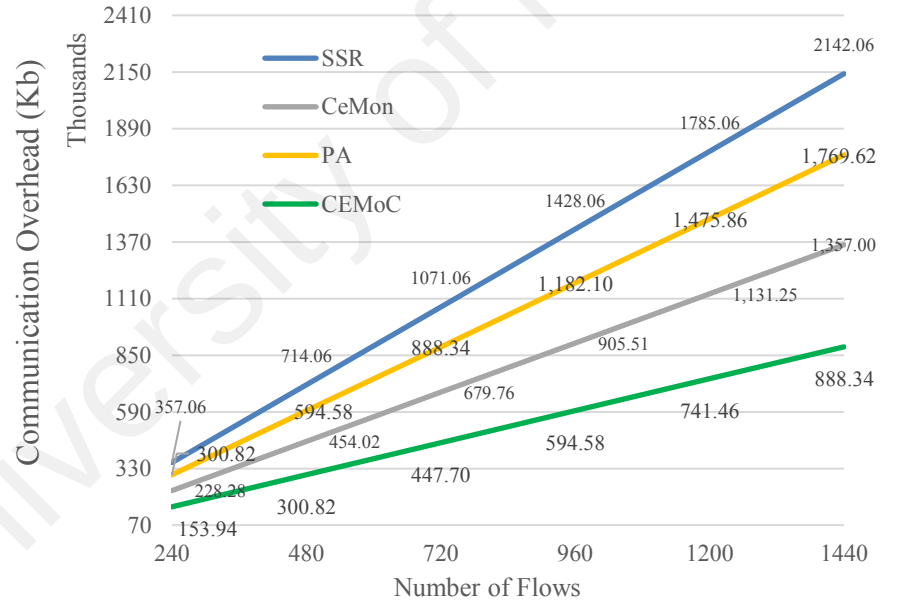


Figure 5.8: Total Communication overhead with one Controllers

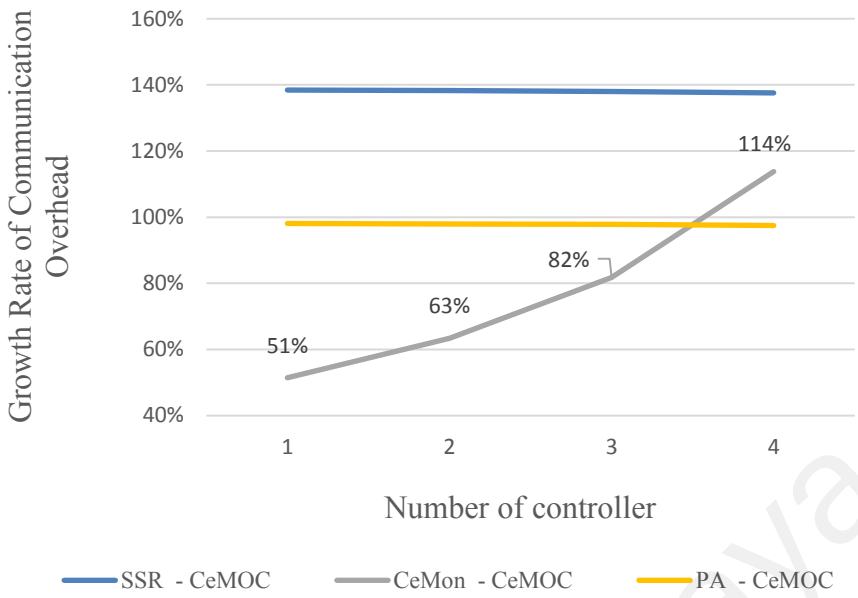


Figure 5.9: Average Growth Rate of Communication Overhead with Different Numbers of Controllers

Table 5.3: The Growth Rate of Benchmarks with Different Flows and Controller Numbers over CEMoC.

Benchmark / Flows	240	480	720	960	1200	1440
	Growth rate in 4 controllers [%]					
SSR	130	136	139	140	140	141
CeMon	109	113	114	115	116	116
PA	94	97	98	98	99	99
	Growth rate in 3 controllers [%]					
SSR	131	137	139	140	141	141
CeMon	78	81	82	83	83	83
PA	95	97	98	99	99	99
	Growth rate in 2 controllers [%]					
SSR	132	137	139	140	141	141
CeMon	60	63	64	64	65	65
PA	95	98	98	99	99	99
	Growth rate in 1 controller [%]					
SSR	132	137	139	140	141	141
CeMon	48	51	52	52	53	53
PA	95	98	98	99	99	99

In addition, the number of paths from the controller to the switches is increased as more switches are attached indirectly through in-band data paths. Thus, more switches to poll results in more flow statistics to communicate. Table 5.3 also illustrates the growth

rate of our benchmarks in different flows and controller numbers over CEMoC. It indicates that there is a slight but steady growth of SSR and PA over CEMoC when the number of controllers increases. However, CeMon reports a decrease in the communication overhead when the number of controller decreases. It is also observed that the overhead in all methods grows when the number of controller drops from four to one. Table 5.4 explains the average growth rate in comparison to four controllers. As can be seen, CeMon reports the lowest growth rate by 17%, 33% and 50% in comparison to four controllers. However, other methods result in a higher growth rate, i.e., 37%, 75%, 112 %.

Table 5.4: The Average Growth rate in Comparison to four controllers

Number of controllers / Method	Growth rate [%]		
	3	2	1
SSR	37	75	112
CeMon	17	33	50
PA	37	75	112
CEMoC	37	74	111

5.2.2.2 **Message Interaction Overhead**

The evaluation compares CEMoC's message interaction caused by polling switches to SSR, PA, and CeMon with different flow numbers. Figure 5.10 shows message interaction with four controllers, where CEMoC and PA achieved the most efficient number of message interactions in all of the iterations. However, PA sacrifices other overheads such as communication and controller overhead at the expense of low message interaction. This is because PA aggregates the whole flow stats in the switch in only one message. The efficient number of message interaction, is constant (a fix number), can be found in the methods that only apply polling all flows. This strategy prevents excessive sending and receiving messages from controllers and switches. CeMon is in third place as it can reduce almost half of the iterations using polling all from core switches. SSR

demonstrates the highest number of interactions as a result of polling switches for every individual flow. In total, CEMoC reduces this overhead by up to 28% and 143% over CeMon and SSR, respectively. Table 5.5 shows the result of message interaction with different numbers of controllers. It is observed that methods that apply single polling for every individual flow do not change with different numbers of controllers. However, CEMoC and PA, which apply polling all flows, achieve a slight decrease when the number of controllers is reduced.

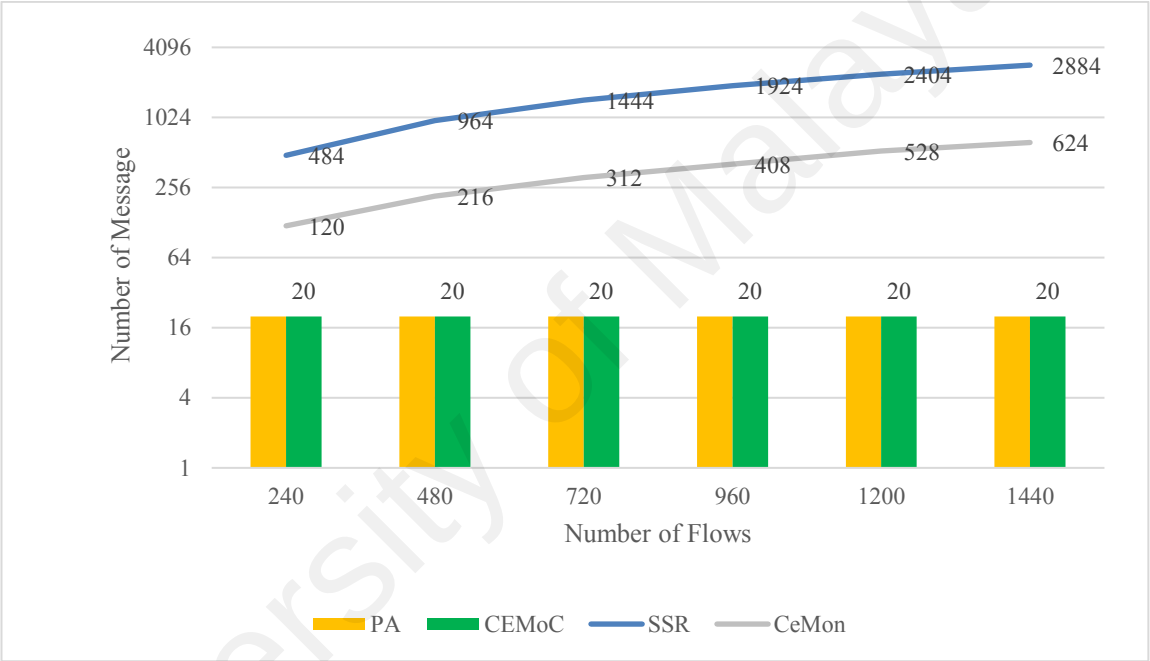


Figure 5.10: Message Interaction in 4 controllers

Table 5.5: Message Interaction with Different Numbers of Controllers

Benchmarks / Flows	Number of Message Interaction					
	240	480	720	960	1200	1440
	Message Interaction in 3 Controllers					
SSR	483	963	1443	1923	2403	2883
CeMon	107	203	299	395	491	587
PA	19	19	19	19	19	19
CEMoC	19	19	19	19	19	19
	Message Interaction in 2 Controllers					
SSR	482	962	1442	1922	2402	2882
CeMon	106	202	298	394	490	586
PA	18	18	18	18	18	18

Table 5.5, Continued

CEMoC	18	18	18	18	18	18
	Message Interaction in 1 Controller					
SSR	481	961	1441	1921	2401	2881
CeMon	105	201	297	393	489	585
PA	17	17	17	17	17	17
CEMoC	17	17	17	17	17	17

5.2.2.3 **Controller Overhead**

Figure 5.11 explains the total overhead of four controllers with different flow numbers. It is observed that the controller overheads of all methods are linearly increased at the expense of flow growth. Thus, the more flows are counted, the more overhead is impressed. Unlike PA and CeMon, which report all the flow stats for calculation, CEMoC only reports the required statistics, which contributes to the utilisation of UDP flows. As a result, there are averages of 99%, and 126% of reduction in the controller overhead by CEMoC over PA. However, SSR reports the flow stats in the same way as CEMoC. The difference between SSR over CEMoC is that SSR generates and reads a massive number of files with regard to the number of flows, as every flow is placed in one file. Nevertheless, as CEMoC aggregates only required stats in one file, there is only one file to read. CEMoC saves up to 65% of controller overhead over SSR. It is also observed that the total controller overhead is reduced by less than 0.01% on average when the number of controllers is decreased. Table 5.6 reports the total controller overhead with different numbers of controller. As can be seen in the table, there is a negligible reduction in overhead when the number of controllers drops to one.

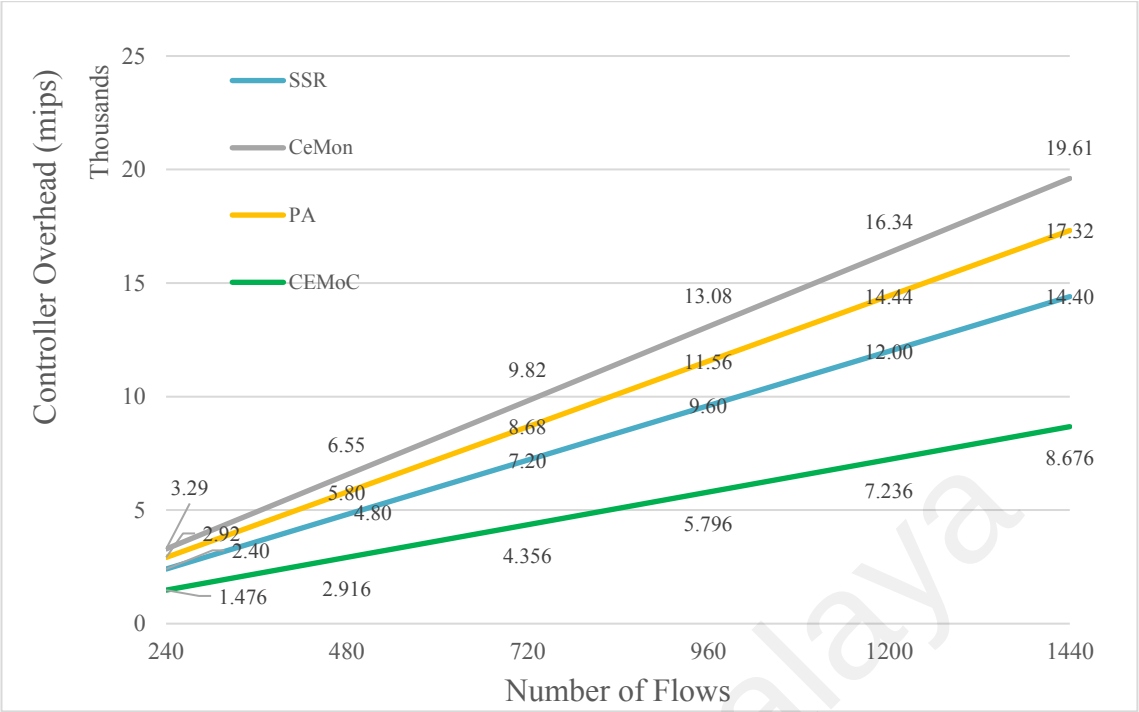


Figure 5.11: Controller Overhead in four Controller Scenarios

Table 5.6: Controller Overhead with Different Numbers of Controllers

Benchmarks / Flows	Overhead in different flow number					
	240	480	720	960	1200	1440
	Controller overhead in 3 Controllers					
SSR	2403	4803	7203	9603	12003	14403
CeMon	3286	6550	9814	13078	16342	19606
PA	2915	5795	8675	11555	14435	17315
CEMoC	1475	2915	4355	5795	7235	8675
	Controller overhead in 2 Controllers					
SSR	2402	4802	7202	9602	12002	14402
CeMon	3284	6548	9812	13076	16340	19604
PA	2914	5794	8674	11554	14434	17314
CEMoC	1474	2914	4354	5794	7234	8674
	Controller overhead in 1 Controller					
SSR	2401	4801	7201	9601	12001	14401
CeMon	3282	6546	9810	13074	16338	19602
PA	2913	5793	8673	11553	14433	17313
CEMoC	1473	2913	4353	5793	7233	8673

5.2.2.4 Accuracy in Multiple-controller (distributed controller) with in-band deployment

Unlike statistical estimation models or sampling methods used in traditional networks, accuracy mainly corresponds to time. Basically, due to network latency and the sequential creation of messages in the controller, synchronising poll requests are infeasible for all the switches in a network. Also, the exact moment of reading flow counters in the switches is unknown (Megyesi et al., 2017). As a consequence, estimating flow utilisation may be limited by a negligible error rate. This problem is also referred to as *Accuracy limitation for lack of synchronisation,* which can be more sophisticated when dealing with in-band deployment where statistic request and result traverse through the network's data plane paths. The experiment conveyed a 360-second experiment with a VBR traffic pattern to highlight the observed error and the impact of different controller numbers on accuracy.

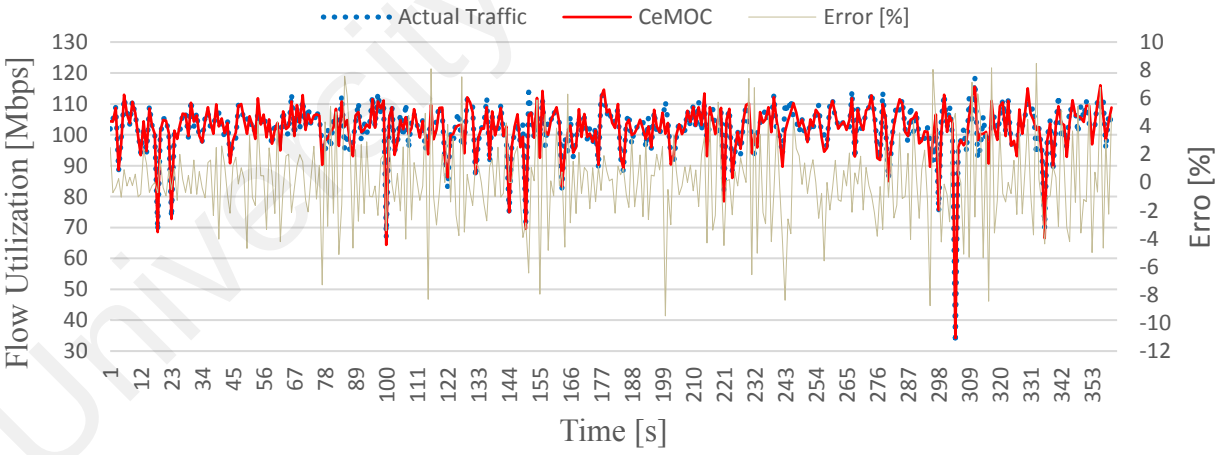


Figure 5.12: Actual measured flow utilisation captured by Wireshark, CEMoC and the relative error.

Figure 5.12 shows the actual measured flow utilisation captured by Wireshark, CEMoC and the relative error with four controllers and no extra delay. It can be observed that the flow utilisation captured by CEMoC is very close to the actual one. In fact, CeMon reports a maximum absolute error and standard deviation of 9.49% and 1.98%,

respectively. Artificial delays were introduced in the network to understand the impact of different numbers of controllers on the accuracy of flow statistical measurements.

Table 5.7 illustrates the relation between error ratio with different number of controller and delays. It can be observed that the error ratio increases in different number of controllers when the delay increases. Thus, the more delay is introduced, the bigger the error ratio. However, based on Table 5.7 it is concluded that the number of controllers has a direct relationship to the error ratio, by which the error ratio for all delays decreases when the number of controllers increases and vice versa.

In order to construct a full traffic matrix for collecting all flow stats from all controllers, all controllers send the counted flow bytes (stats) and aggregate them into a UDP file. The controllers then send UDP files to the coordinator. It should be recalled that all transmissions in in-band deployment take place through the data plan network links. Therefore, in the worst case a packet in a fat-tree topology may go through five switches and links, each of which may impose different delay to the packet, until it reaches the coordinator.

Table 5.8 shows the maximum transferring delay of the final UDP packet from each controller to the coordinator. Therefore, in the worst case, CEMoC is able to record flow utilisation without overlapping in the next intervals. However, the corresponding flow entry was proactively installed to the switches to transfer the UPD packet from the controllers to the coordinator. A static flow entry from controllers to coordinator was set in every switch in the network to eliminate the delay introduced in Table 5.8.

Table 5.7: The Relation between error ratio on different controller numbers and delays.

Error / Delay [ms]	Error [%]				
	0	5	10	15	100
	Relative Error with 4 Controllers				
Max Absolute Error	9.48E-02	9.46E-02	9.43E-02	9.36E-02	9.00E-02
	Relative Error with 3 Controllers				
Max Absolute Error	9.48E-02	9.44E-02	1.01E-01	1.62E-01	4.64E-01
	Relative Error with 2 Controllers				
Max Absolute Error	9.47E-02	9.43E-02	1.17E-01	1.95E-01	5.86E-01
	Relative Error with 1 Controllers				
Max Absolute Error	9.46E-02	1.00E-01	1.32E-01	2.28E-01	7.08E-01

Table 5.8: Maximum transferring delay of final UDP packet from each controller to the coordinator.

Network Delay [ms]	0	5	10	15	100
Controller to Coordinator [ms]	14.2	39.7	68.9	159.4	606.6

5.2.3 Experiment III: Simulation: multiple-controller with in-band deployment

This experiment aims to evaluate the performance of the proposed framework in in-band network deployment, under a real dataset workload and the 2-tire fat-tree topology presented in section 5.1.3 and 5.1.4 respectively. Due to the large scale of the experiment (more than 100,000 flows; the flows' interarrival time is between 4ms and 40ms for 80% of the flows) the experiment was conducted in a trace-driven simulator. The experiment only simulates the event of flow arrival and expiration in the network, and only focuses on various overheads, as the evaluation of accuracy requires real specifications and behaviours of the network, i.e. jitter and different latencies. The observed findings of CeMon (Su et al., 2015) were advertently omitted, because the dataset only provides edge switches. Therefore, due to the identical behaviour of PA and CeMon when using only edge switch(es), the finding from PA represents the CeMon as well. In this simulation, the minimum and maximum number of controllers is set to one and nine respectively. For

better understanding, the findings of the experiment are first depicted with nine controllers. Then the section summarises the findings from different numbers of controllers and illustrates them as the total overhead in a 60-second trace. The following elaborates the performance evaluation of the proposed framework with multiple controllers and in-band network deployment.

5.2.3.1 **Communication Overhead**

Figure 5.13 explains the simulation results of communication overhead with nine controllers. Findings from the simulation experiment confirm the results observed in other experiments. It is observed that CEMoC steadily achieved the best performance in terms of decreasing communication overhead in all experiments. However, SSR and PA change places in several points of time. Findings show that PA achieves better performance than SSR whenever the total number of flows is twice the number of demanded flows. This behaviour of PA can be found in the seconds 15, 19 and 31 onward of the time axe.

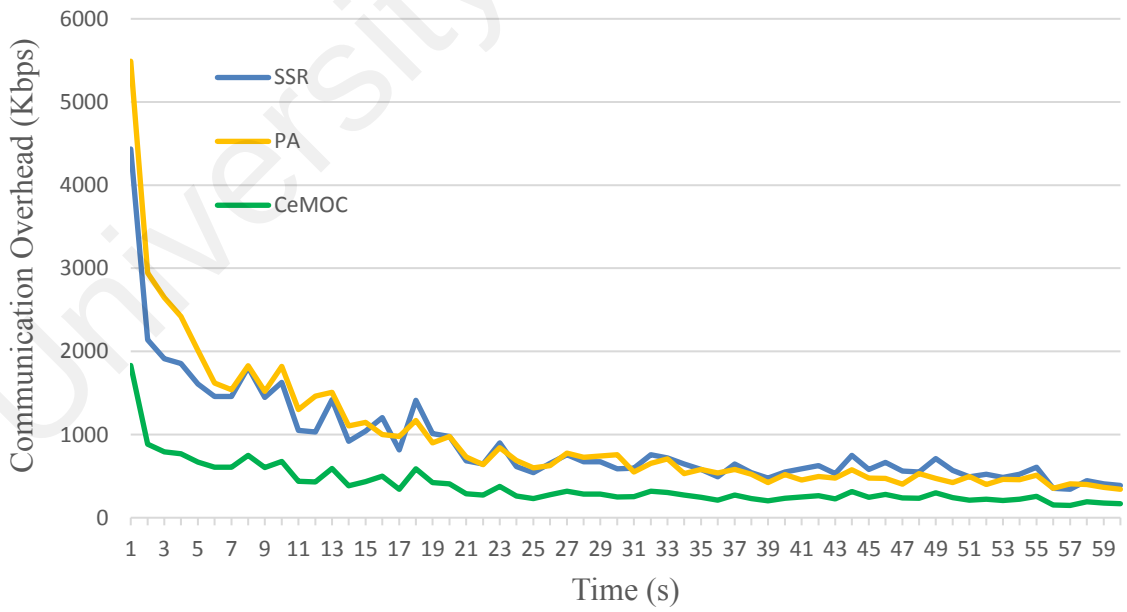


Figure 5.13: Communication overhead with nine controllers

Findings also show a meaningful decrease when the number of controllers increases. Figure 5.14 explains the total communication overhead with different numbers of

controllers for 60 seconds. As can be seen, there is a sharp fall in all methods when the number of controllers reaches two. This is because all the edge switches in the second pod (i.e. nine edge switches in this topology) send their stats by an extra inter-pod link. This extra inter-pod link decreases the communication overhead by roughly 9%. The depression declines steadily to the end (9th controller). In Figure 5.14 it can be observed that in particular CEMoC outperforms PA and SSR by 153% and 140%, respectively.

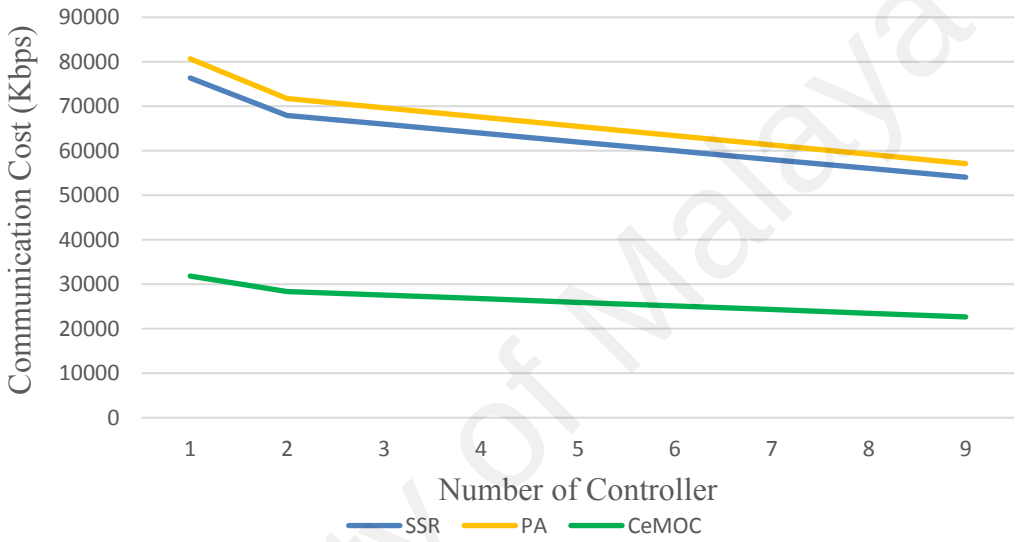


Figure 5.14: Total communication overhead with nine Controllers for 60 seconds

5.2.3.2 **Message Interaction Overhead**

Figure 5.15 depicts the message interaction overhead from the 60-second trace of the experiment. Findings from this section confirm the experiments via emulation in the earlier section and our problem formulation in Chapter 3. As witnessed in Figure 5.15, CEMoC achieves the optimal number of message interactions by an average of 45 messages per second. PA reports a near optimal number, however, it causes abnormalities in several periods in the time line (i.e. 0s-6s, 8s, and 10s). This behaviour is related to the multipart OpenFlow message, as comprehensively explained in Chapter 3. It should be recalled that the length of the TCP packet in the wire medium cannot exceed 64K. Therefore, PA shows message number three times more (3x more) than CEMoC in the

mentioned time periods, as a result of exceeding packet size from the standard predefined value 64Kb. Through examining and tracing with Wireshark, it was observed that whenever the number of flow entries exceeds 453, the reply packet is divided into two or more messages. SSR results in the highest number of messages by generating three messages for every single flow in a second. In addition, similar to the emulations, findings from the simulation show the increase of message interaction at the expense of controller numbers in which the number of messages is increased when the number of controllers is increased.

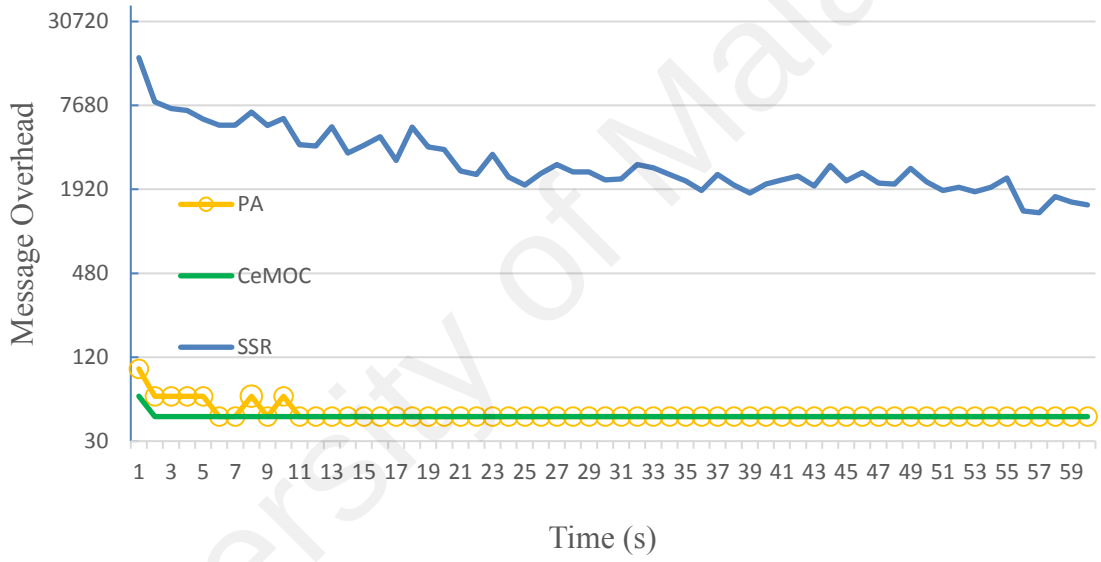


Figure 5.15: Message Interaction overhead with nine Controllers in 60 seconds.

Table 5.9 reports the message interaction overhead with different numbers of controllers in 60 seconds. According to the table, CEMoC and PA show the biggest change, while SSR achieves the smallest change with 0.029%. However, the standard deviation of changes for all methods uses the identical ratio (i.e. 164.31), that indicates that the same changes ratio to the proportion of total number of message for each method. It means, the ratio of changes among SSR, PA, and CEMoC are exacly the same, though, the average of difference is different. In simple words, different numbers of controllers

make no difference to the change ratios. In total, the proposed method showed a 6% and 63% reduction in message interaction overhead over PA and SSR, respectively.

Table 5.9: Total Message interaction overhead with different numbers of controllers in 60 seconds.

Number of Controller	Method		
	SSR	PA	CEMoC
1	205476	2220	2220
2	205536	2280	2280
3	205596	2340	2340
4	205656	2400	2400
5	205716	2460	2460
6	205776	2520	2520
7	205836	2580	2580
8	205896	2640	2640
9	205956	2700	2700
Average of difference	0.029%	2.32%	2.50%
STD of changes	164.31	164.31	164.31

5.2.3.3 **Controller Overhead**

Similar to the emulation findings in the controller overhead, the findings from the simulation demonstrated similar results and behaviour. CEMoC could save overheads up to 65% and 154% as compared to SSR and PA, respectively. Figure 5.16 plots the controller overhead in 60 seconds with nine controllers. It is observed that the overhead trends of all methods remained the same during 60 seconds. As can be seen, CEMoC and SSR stay in a close range of each other until the end of the experiment. However, PA starts and finishes the journey relatively far from other methods mentioned above. This is because PA transfers a huge amount of statistics in a message (every time), which is the total active flows at the moment, and this vast information needs to be processed; CEMoC and SSR transfer only the demanded flows to be processed. The observed results also confirm the finding from the emulation in which there is a negligible reduction in

overhead when the number of controllers reduces. Figure 5.17 depicts the total overhead during 60 seconds.

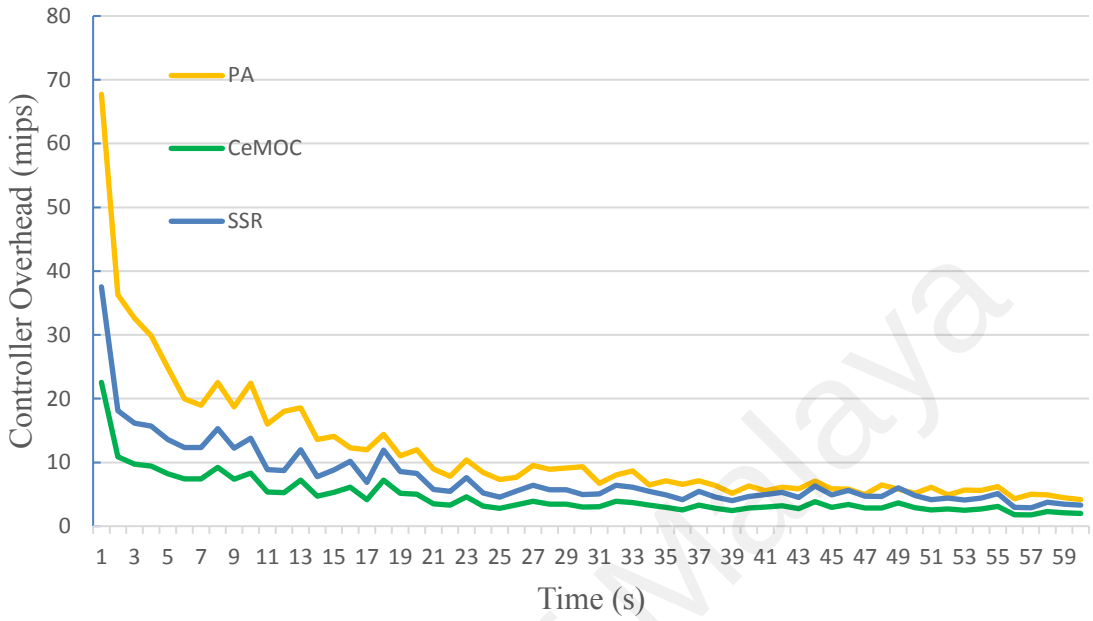


Figure 5.16: Controller Overhead with nine controllers in 60 seconds

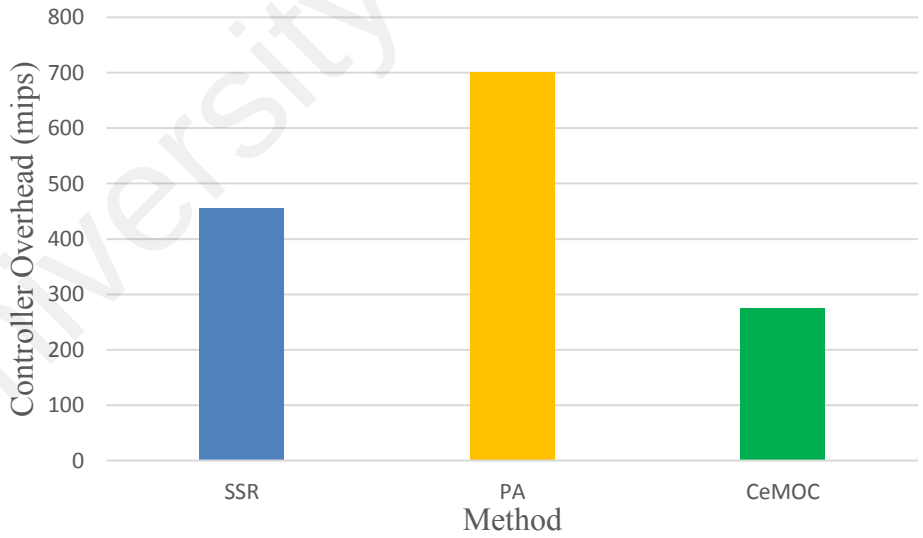


Figure 5.17: Total controller overhead with different numbers of controllers in 60 seconds.

5.3 Statistical Modelling

This section seeks to verify the findings collected from the simulation experiment. To address this objective, the section applies a statistical analysis model to approximate the correlation coefficient of the samples (the findings from the emulation and simulation) and the significance of the differences or similarities. A t-test is used as the statistical analysis modelling tool to determine the significance of the differences. To reach the aim of this section, a t-test with equal variances was chosen due to the samples' limitation (n < 30) where n is the number of the population in each sample. The main aim of the t-test is to either accept or reject a given hypothesis (i.e. "There is statistically a significant difference between two sets") by a probability value $0 \leq p \leq 1$, a confidence level = 95% and a significance level $\alpha \leq 0.05$ which is the most widely used and considered to be "small enough" (Zar, 2013). Therefore, the given H_0 is accepted if the probability p is less than the significant level α. Another supportive approach that determines whether to accept or reject the H_0 is through calculating the t-value and comparing it to the critical value. Thus, if the t-value is higher than the critical one, the H_0 is rejected. The employed critical value in our t-test for paired and unpaired analysis are 2.570 and 2.228, respectively. A full list of the critical values of the t-distribution is shown in Appendix C, Table 1. For more simplicity and clarity, the section only presents the test findings from CEMoC. The difference between the emulation result and the simulation on the message interaction overhead is the absolute zero value. This is because the findings from the emulator report the exact amount as simulation. Therefore, the findings of the t-test on message overheads are not depicted, as the t-test generates errors when the subtraction of means and score is zero (i.e. error: division by zero). The test findings for other benchmarking methods are shown in Appendix C.

To verify the correctness of the findings from the simulation, the same input (section 5.2.2, experiment 2) with the given topology and dataset workload (i.e. 4-pod fat-tree and

CBR respectively) was given to the simulator. Thereafter, t-test was used to analyse the two population means and testing the difference between the samples. In the t-test, two means (average) are compared and the possibility of differences in two populations is distinguished. It also notes the significance of the differences. In other words, the t-test defines if the differences could have occurred by chance. Therefore, the findings from the simulator and the emulation were analysed to confirm the correctness of the simulator.

To calculate the t-value on the paired and unpaired test, assuming equal variances and Pearson-correlation ratio (*r)*, equation 5.1, 5.2 and 5.3 are used respectively:

$$t = \frac{(\sum D)/N}{\sqrt{\frac{\sum D^2 - (\frac{(\sum D)^2}{N})}{(N-1)(N)}}} \qquad 5.1$$

$$t = \frac{\overline{x}_1 - \overline{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \qquad 5.2$$

$$r = \frac{N\Sigma xy - (\Sigma x)(\Sigma xy)}{\sqrt{\left[N\Sigma x^2 - \Sigma x^2\right]\left[N\Sigma y^2 - \Sigma y^2\right]}} \qquad 5.3$$

where *t* measures the size of the difference relative to the variation in the sample data. In other words, it calculates the difference represented in the units of standard error. *r* is the Pearson-correlation that measures the correlation between sets of data and defines how well they are related. Table 5.10 explains the annotations used in mean and variance equations. Equations 5.4 and 5.5 show the equations of mean and variance respectively:

$$\overline{x} = \frac{\sum_{i=1}^{N} X_i}{N} \qquad 5.4$$

$$s^2 = \frac{\sum(X_i - \overline{x})^2}{N} \qquad 5.5$$

Table 5.10: Annotation in mean and variance equations.

Annotation	Definition
t	T-Value
r	Pearson Correlation
D	difference between matched scores
N	number of samples inside the set/number of pair of scores
$\sum x$	Sum of x scores
$\sum y$	Sum of y scores
$\overline{x}$	Mean of the set
s^2	Variance of the set

5.3.1 **Communication Overhead**

Tables 5.11 and 5.12 explain the analysis of the emulation and simulation findings for communication overhead of CEMoC in the paired and unpaired t-test.

Table 5.11: Paired t-test Two Sample for Means of communication overhead in CEMoC

	Simulation	Emulation
Mean	246356	246116
Variance	16721510400	16677796147
Observations	6	6
Pearson Correlation	**0.99**	
Hypothesized Mean Difference	0	
Degree of freedom	5	
t Stat	**1.076**	
P(T<=t) one-tail	0.092	
t Critical one-tail	**2.015**	
P(T<=t) two-tail	**0.102**	
t Critical two-tail	**2.570**	

As can be seen in Table 5.11, the t-stat is smaller than the t-critical two-tail value (1.076 < 2.507). It also reports that the two-tail p-value (0.102) is not less than $\alpha = 0.05$.

Thus, the test fails to reject the null hypothesis (H_0). It is concluded that data from two populations support H_0. Therefore, there is no statistically significant difference found between the two given sets (i.e. emulation and simulation). Moreover, the correlation value (r) exposes a strong positive relation (0.978) between the two sets.

Table 5.12: Unpaired t-test Two Samples Assuming Equal Variances of communication overhead in CEMoC

	Simulation	Emulation
Mean	246356	246116
Variance	16721510400	16677796147
Observations	6	6
Pooled Variance	16699653274	
Hypothesized Mean Difference	0	
Degree of freedom	10	
t Stat	**0.003**	
P(T<=t) one-tail	0.498	
t Critical one-tail	1.812	
P(T<=t) two-tail	**0.997**	
t Critical two-tail	**2.228**	

Additionally, the findings from the unpaired t-test in Table 5.12 support the H_0 as the t-stat by 0.003 is smaller than t-critical two-tail value by 2.228. Moreover, the p-value by 0.997 is much higher than the $\alpha = 0.05$. It is therefore concluded that the data from emulation and simulation are considered equal.

5.3.2 Controller Overhead

Tables 5.13 and 5.14 explain the analysis of the emulation and simulation findings for communication overhead of CEMoC in the paired and unpaired t-test.

A similar analysis in communication overhead in section 5.3.1 and 5.3.2 was used to reject the H_0 for this overhead as well.

If t-stat is smaller than the two-tail value, the H_0 is not accepted. As can be seen in the paired test, -0.925 is less than 2.570, and in the unpaired test 0.925 is less than 2.228.

Table 5.13: Paired t-test Two Sample for Means of controller overhead in CEMoC

	Simulation	Emulation
Mean	4996	5071
Variance	6881280	7238626.8
Observations	6	6
Pearson Correlation	**0.99**	
Hypothesized Mean Difference	0	
Degree of freedom	5	
t Stat	**-0.925**	
P(T<=t) one-tail	0.198568395	
t Critical one-tail	**2.015048373**	
P(T<=t) two-tail	**0.397**	
t Critical two-tail	**2.570**	

Table 5.14: Unpaired t-test Two Samples Assuming Equal Variances of controller overhead in CEMoC

	Simulation	Emulation
Mean	4996	5071
Variance	6881280	7238626.8
Observations	6	6
Pooled Variance	7059953.4	
Hypothesized Mean Difference	0	
Degree of freedom	10	
t Stat	**-0.048**	
P(T<=t) one-tail	0.480	
t Critical one-tail	1.812	
P(T<=t) two-tail	**0.961**	
t Critical two-tail	**2.228**	

If the two-tail p-value is no less than α (0.05), it is concluded that data from two populations support H_0. The findings report the two-tail p-value in the paired test is 0.397, which is greater than 0.05. In the unpaired test it is 0.961, which is not less than 0.05 as well.

The correlation value reports a positive large value (0.981), which defines a strong and positive relationship between two sets.

Therefore, it is concluded that there is no statistically significant difference found between the two given sets, i.e. emulation and simulation.

5.4 Discussion

This section provides a discussion of the results and findings from the extensive experiments in the previous sections. All findings are discussed and divided into three overhead categories (i.e. communication, message interaction and controller overhead).

5.4.1 Communication overhead

The findings from experiment 1, 2, and 3 were compared on different deployment models such as out-of-band and in-band. The results showed a significant improvement of the proposed method (CEMoC) over all benchmarking methods. Regardless of network deployment, the findings from the experiments show a linear increase in all of the methods in the proportion of flow numbers in the switch, in which the overhead increases when the flow numbers increase. CEMoC achieved the best performance as it only requests demanded flow statistics. From the experiments it can be observed that PA and CeMon report the same overhead due to requesting all the active flows in the switch.

The experiments in in-band deployment model were replicated with different numbers of controllers. A meaningful improvement on overhead from CEMoC was observed when the number of controller increases. PA performed consistently in both experiment 1 and 2, in which it imposed the highest overheads of all methods.

The results from the simulation in a real dataset confirm the findings from experiment 2 where CEMoC achieves the best savings in communication overhead. However, it shows that PA and SSR can perform in a competitive way, with PA performing better than SSR when the total number of flows in a switch is less than the

number of actual demanded flow times to 2 *(total number of flow in switch < (actual demanded flow x2)*).

5.4.2 **Message interaction overhead**

The message interaction overhead in the network was also evaluated under different experiments. The findings from in-band deployment showed that the proposed method exposes the optimal overhead of message interaction on the network. Although CEMoC achieves the same number of message interactions as PA, this confirms that other methods are unable to overcome the proposed method on this overhead. CeMon also performed like CEMoC and PA in this experiment. It was also observed that the overhead of CEMoC remains the same in any arbitrary flow number. In general, the proposed method showed a 63% reduction over CeMon.

In addition, results from the out-of-band deployment model describe same behaviour. However, CeMon emerged as a suboptimal method when there is more than one switch for pulling purposes. The proposed method showed the optimal number of message interactions for all flows, which is the same as the PA result. However, all methods showed that their overheads increase when the number of controllers increases. The relation of message interaction overhead to the number of flow has a contrary behaviour to the communication overhead, where an increase of the number of controller cases a decrease in the communication overhead.

The results from experiment 3 also confirm the findings from previous experiments where the overhead of message interaction in the network is increased at the expense of increasing the number of controllers. It is observed that CeMoC outperforms PA when the number of messages in a switch is high. As PA requests all messages in a switch, it revealed suboptimalitywhen the number of flow increases and causes the

reply packets divides to multiple packets. However, CeMoC results in the same message number when the number of flows is relatively low.

5.4.3 **Controller overhead**

Similar to all other overheads, different experiments were evaluated for the overhead of controllers as well. In out-of-band deployment (experiment 1), the proposed method reported the highest saving in this overhead. However, similar to communication overhead and message interaction, PA and CeMon show the same overhead.

In the in-band deployment model (experiment 2 and 3) where multiple controllers were employed, the proposed method showed a significant saving over all other benchmarking methods. In total CEMoC performs up to 65% better than SSR. However, the most significant achievement is the comparison of the proposed method with CeMon and PA, in which CEMoC outperforms the two mentioned methods by 99% and 126%, respectively. Moreover, the evaluation of the controller overhead was followed by comparing the methods with different numbers of controllers; here it showed a negligible decrease in the total overhead when the number of controllers increases. However, the findings of multiple controllers show the total overhead on all controllers, where the overhead is divided by the number of controllers to observe the overhead on each individual controller.

In addition, the result of the flow measurement from our proposed method in in-band deployment was captured and shown with different controller numbers and various imposed delays in the network. It was observed that the error ratio increases as the number of controllers decreases. This is due to the number of links and switches in the network. With four controllers in the network, there will be fewer links and statistics for a switch to reach the corresponding controller. Therefore, the lower the

number of controllers, the more links and switches the statistics go through. The situation deteriorates when the imposed delay is extended.

5.4.4 Significance of Evaluation

Beside the main findings, the following lessons can be inferred through evaluation of the performance in the simulation and emulation, and the analysis and comparison of the network with different deployment models (i.e., out-of-band and in-band):

- The in-band deployment network model is costly in comparison to the out-of-band model in terms of the generated overhead by the pulling of statistics of flows.
- Employing multiple controllers in out-of-band deployment does not have any impact on the generated overhead.
- Employing multiple controllers can significantly decrease the communication costs as well as the controller overhead in in-band deployment.
- Using multiple controllers in the network distributes message interaction among all controllers. However, in total, it increases a negligible overhead on message interaction in the network.
- Implementing multiple controllers distributes the load on several controllers, and hence significantly minimises the overhead on each controller. However, the total overhead is increased by a negligible cost.
- Different network deployment models leave no impact on the overhead imposed on the controller(s).
    - Accuracy of flow statistical measurement is improved when employing multiple controllers in in-band and out-of-band network deployment.

5.5 Summary

This chapter presented the performance evaluation of the experiments (i.e. emulation and simulation) of the proposed framework with their corresponding counterparts. The aim of elaborating the evaluation in different experiments is to explore the unique objective and significant findings from each experiment. The chapter comprehensively explained the evaluation setup accompanied by its relative components (i.e. dataset, topology, performance metrics, and benchmarking methods). It also conducted three experiments and performed evaluations to demonstrate the outperformance of the proposed method compared to the state-of-the-art methods, and discussed the findings and results inside each performance metric. Finally, the chapter performed and explained a statistical analysis test through paired and unpaired t-tests for verification and validation of the results from our simulator.

CHAPTER 6: CONCLUSION

This chapter is the epilogue of the research carried out in this study. It summarises the main findings and the attained research objectives, and highlights the significance of the proposed method. The chapter also explains the limitations of this study and suggests future extensions. The chapter is organised as follows:

Section 6.1 discusses the reappraisal of the main findings and research objectives. Section 6.2 highlights the contribution and achievement of the study. Section 6.3 presents the limitation of the study. Section 6.4 explains the future directions and possible extensions.

6.1 Research questions and research objectives

Next generation networks are characterised by their huge scale and the diversity of the generated traffic. In such networks, various real-time (latency-sensitive) and offline applications require information on the current state of network traffic and its behaviour. This necessary requirement cannot be accomplished without enough measurement of data about individual flows in each part of the network. The main motivation of this thesis is to address the problems of network traffic flow measurement systems in SDN due to demanding features for other network applications. In addition, due to the various implications of measurement systems such as different overheads, accuracy, and being real-time in SDN, these implications can potentially be seen as a unified multi-objective framework.

This study aimed to achieve a multi-objective network flow measurement system that effectively minimises multi-objective costs such as communication, message interaction, and controller overheads in near real-time, with high accuracy in both centralised and decentralised controller scenarios in DCN. Section 1.4 details the four research objectives

of this study. Therefore, this section aims to answer the following questions: a) RQ1: What existing flow measurement systems for networks are there? b) RQ2: How can the behaviour of SDN flow measurements be formulated? c) RQ3: How can a flow measurement system be developed in a near real-time and cost-effective manner? d) RQ4: what is the effectiveness of the proposed framework when compared to existing approaches?

Objective 1: To study the traditional network traffic measurement and monitoring approaches and perform a gap analysis review on the state-of-the-art SDN techniques for network traffic measurement and monitoring.

The first objective is provided to answer RQ1 of this study. To accomplish this objective, a thorough discussion was first conducted on major representative research in the area of network traffic measurement. The objective was to broadly review network traffic monitoring/measurement implications and introduce traditional measurement and monitoring methods for network traffic. Two major approaches were presented for measurement/monitoring purposes, namely the active and passive approach, which are able to accomplish different monitoring tasks. An overview of SDN was presented to introduced different layers and the architecture alongside the underlying fundamental concept, to help readers gain an easy and smooth understanding of SDN. Moreover, a light-weight overview of the original SDN measurement approaches was introduced to OpenFlow specifications, namely SSR, PA, push-based, and trigger-based. Finally, the study discussed the state-of-the-art SDN measurement solution and the latest trends in flow-based network traffic measurement in SDN in detail.

Objective 2: To propose a comprehensive mathematical formulation and analysis on different costs such as communication overhead, message interaction, and controller

overhead as a multi-objective problem in the context of network traffic flow measurement.

This objective aimed to address RQ2 to provide a clear understanding of various costs associated with the measurement of flow, both on the data and control plane side. Basically, the measurement task imposes overheads on different aspects. These aspects affect the control plane and data plane at the same time. The severity of overheads may vary in different network deployment models. In addition, the objective aimed to analyse the problem that was highlighted in Chapters 1 and 2, and to conduct an in-depth investigation to show the impact of flow measurement on different aspects of overhead. In addition, different network deployment models such as out-of-band and in-band were formulated. This objective focused on three imposed overheads, of which two were highlighted in the literature review, the communication and message interaction overheads imposed by pulling flow statistics. It then introduced a new overhead, which is generated by reading and calculating the measured flows statistics. Finally, a mathematical explanation was formulated for in-band and out-of-band network deployments.

Objective 3: To propose a multi-objective flow measurement framework that effectively minimises the costs and provides near real-time flow statistics in fully centralised and distributed SDN controllers.

This objective addressed RQ3. It was achieved by designing a multi-objective framework that effectively reduces different overheads in the data and control plane. The framework leverages the OpenFlow protocol to provide a real-time active traffic flow measurement. Since the proposed framework adopts the OpenFlow group table feature, it is able to optimally define the demanding flows in every switch, and poll the switch to

read demanding flows statistics. In addition, the framework was designed based on polling edge switches, therefore, it generates a near real-time measurement of flows. Moreover, it implemented a pulling-based approach which has been proven to be more accurate as compared to push-based and passive approaches. Finally, the approximation algorithm *eager-greedy* was adopted to reduce the complexity of the optimisation problem of *weighted set-cover* and the complexity imposed by the *brute forced algorithm*. The approximation algorithm assigns a weight for every switch in order to find the switches with the lowest communication costs to the location of the controller and the coordinator. It also takes delays into consideration for setting the weight of the switches.

Objective 3: To evaluate the performance of the proposed multi-objective framework against similar existing state-of-the-art approaches in SDN.

This objective provided the answer to RQ4 by presenting and discussing the result of the performance evaluation of the proposed framework in detail. First, the evaluation setup was demonstrated, which consists of a detailed description of the performance metrics, benchmarking methods, datasets, and topologies. Second, three experiments were performed, whereby the obtained results from implementing the framework exhibited a linear and consistent trend. The first two experiments were conducted in an emulated environment with artificial datasets, as the emulation limits the machine power and capacity to perform the experiments with a real dataset. Findings from both emulations showed a linear increase associated with the flow number. Also, it was proven that the overheads increase in in-band deployment in relation to the number of flows. In addition, it was understood that deploying multiple controllers affects the overheads, whereby an increased number of controllers reduces the communication overhead and increases the message interaction overhead. The last experiment was performed through a simulator using a real dataset. In addition, a comprehensive discussion exposed the

outperformance of the proposed framework in comparison to other state-of-the-art work. The findings from this experiment confirm the results obtained in previous experiments. The findings from all experiments confirm the linearity and consistency of the trend. Third, the study highlighted statistical modelling for verification of the achieved results from the simulation experiment. Linear regression was performed on all metrics derived from the cost factors and compared to prove the identicality with the emulation results using the t-test. The t-test result showed a significant similarity between the result obtained from the emulation and the result achieved from the simulator. Finally, a thorough discussion of all the experiments and results along with the significance and lessons learnt was presented.

6.2 Achievement of the Study

This study proposed a multi-objective network flow measurement framework in SDN suitable for datacentre networks. In detail, the main achievements of this study are as follows:

i. The major strategies used in the design of traffic measurement and monitoring, namely active and passive approaches, were explored to identify the underlying common concepts and mechanisms of traffic measurement/monitoring. Moreover, a comparison between the mechanisms mentioned above was made.
ii. A comprehensive review of the existing solutions and taxonomy of the approaches in SDN network traffic measurement and monitoring was presented. This review can be used as a comprehensive tutorial for fellow researchers who are interested in this topic.
iii. A comprehensive problem formulation and analysis for cost-effective flow measurement in SDN for in-band and out-of-band network deployment was

performed. This is the first comprehensive mathematical formulation for flow measurement costs to date.

iv. A fine-grain cost-effective framework in SDN was proposed for DCNs, which measures network traffic flow in near real-time manner. The extended proposed framework was presented for a decentralised (multiple) controller scenario. To the best of the researcher's knowledge, this work was the first attempt to design a traffic measurement system for a decentralised controller scenario.

v. Several rigorous evaluations of the proposed framework were performed in an emulation environment and simulation. The emulation was done through Floodlight, a popular java-based SDN controller, to emulate the SDN setting Mininet was employed.

vi. A precise and trusted simulator was designed and developed for the purpose of evaluation on a large network scale with the real dataset. The simulator was tested using statistical modelling/testing and proved to be promising.

vii. Finally, a large public network traffic dataset was adopted from a university datacentre and converted to excel (.xsl). This dataset can be used by fellow researchers who are interested in implementing a real dataset and evaluating their proposed work in simulators.

6.3 Limitations of the study

Individual research typically encounters some limitations in different stages of the project. Several limitations were faced during the implementation and evaluation of this study, and some avenues that may have been of interest are explored in this section to provide future researchers with some lessons and suggestions, in order to enable them to better manage their work. The limitations of the study can be summarised as follows:

(a) Implementation of a real DCN. The study employs OVS (open vSwitch) to emulate the behaviour of an SDN switch. As no such switch was available, the implementation phase was conducted in an emulated environment. Although the emulation is realistic and very close to real environment behaviour, it still cannot be considered identical.

(b) End-host shortage. Due to lack of machines in the lab environment, the study emulates end-hosts connected to the OVS by Linux kernel using Mininet emulator. In addition, the capacity of virtual links was limited due to hardware limitations, as virtual links in Linux consume RAM for emulation purposes.

(c) Lack of machine power. Due to the limitation in machine power and hardware restrictions such as CPU and RAM for large-scale implementation in experiment 2, the study implements the proposed framework in an instance of Amazon EC2.

(d) Experimental evaluation. The evaluation of the performance metrics is carried out through emulations. However, mathematical calculations were used for calculating CPU utilisation (controller overhead), as the CPU workload and instructions (i.e. *I/O* and *calculation)* cannot be separately reported for each thread and task. Although there are a number of benchmark tools to report CPU utilisation, they cannot report on each individual thread with separated workloads.

(e) One of the main objectives of the study was to minimise the costs of flow measurement. However, the study did not address the related costs (overhead) in the network device as the cost comprises different elements and overheads in the device such as TCAM, CAM, CPU, and Buffer which is out of scope of this research. In addition, measuring the overhead on the switch requires the real device to evaluate its performance. However, this study was implemented in mininet that is an emulation tool to emulate the behaviour of the network.

6.4 Suggestion for Future Work

Although this study achieved all its defined objectives, it offers a few directions and suggestions for future research. This section presents possible future research directions that can be pursued to extend SDN measurements and monitoring systems.

(a) QoE and QoS-based IoT traffic monitoring. As an emerging global internet-based information architecture, the Internet of Things offers a dynamic global network infrastructure by embedding intelligence into the environment. This new concept connects everyday objects (Gubbi et al., 2013), every time and everywhere. Today, with the ever-increasing volume of data and network traffic, Quality of Experience (QoE) and Quality of Service (QoS) are the crucial desires in the eyes of internet service providers (ISP) and datacentres (Robitza et al., 2017). Quality monitoring and real-time measurements can play a vital role in the first step of providing high-quality services. In addition, the optimal selection and composition of services is a crucial requirement for ensuring QoS from the users' perspective (White et al., 2017). Real-time monitoring and measurement can be a solution with a high potential to guarantee this vital requirement.

(b) A real-time measurement and monitoring framework for the reduction of energy consumption in edge. With the ever-increasing IoT devices and edge services, the volume of traffic in the edge is increasing exponentially. This massive volume of traffic requires pervasive and cost-effective monitoring in real-time, as analytical tasks and applications can be performed on the edge nodes, which can also help to decrease the energy consumption of datacentres (Patel & Pandya, 2017).

(c) An accurate and real-time DDOS detection and mitigation framework. A sudden massive amount of traffic targeted to a single node has a catastrophic impact. Associated service(s) or nodes can also become partially or totally unavailable (Rebecchi et al., 2017). An accurate and real-time monitoring capability offered by

SDN can effectively detect suspicious traffic (flow/packet-level) and potentially mitigate the attack. Moreover, by employing the real-time monitoring features offered by SDN, sophisticated anomalies can be effectively detected in the entire network (Lee et al., 2017)

REFERENCES

Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2014). A roadmap for traffic engineering in SDN-OpenFlow networks. *Computer Networks, 71*, 1-30.

Azodolmolky, S. (2013). *Software Defined Networking with OpenFlow*: Packt Publishing Ltd.

Bandi, N., Metwally, A., Agrawal, D., & El Abbadi, A. (2007). *Fast data stream algorithms using associative memories.* Paper presented at the Proceedings of the 2007 ACM SIGMOD international conference on Management of data.

Bar-Yossef, Z., Jayram, T., Kumar, R., Sivakumar, D., & Trevisan, L. (2002). *Counting distinct elements in a data stream.* Paper presented at the International Workshop on Randomization and Approximation Techniques in Computer Science.

Bari, M. F., Boutaba, R., Esteves, R., Granville, L. Z., Podlesny, M., Rabbani, M. G., . . . Zhani, M. F. (2013). Data center network virtualization: A survey. *IEEE Communications Surveys & Tutorials, 15*(2), 909-928.

Benson, T., Akella, A., & Maltz, D. A. (2010). *Network traffic characteristics of data centers in the wild.* Paper presented at the Proceedings of the 10th ACM SIGCOMM conference on Internet measurement.

Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., . . . Parulkar, G. (2014). *ONOS: towards an open, distributed SDN OS*. Paper presented at the Proceedings of the third workshop on Hot topics in software defined networking, Chicago, Illinois, USA.

BigSwitchNetworks. (2016). Floodlight v1.2. from https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/24805419/Floodlight+v1.2

Botta, A., Dainotti, A., & Pescapé, A. (2012). A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks, 56*(15), 3531-3547.

Cantieni, G. R., Iannaccone, G., Barakat, C., Diot, C., & Thiran, P. (2006). *Reformulating the monitor placement problem: Optimal network-wide sampling.* Paper presented at the Proceedings of the 2006 ACM CoNEXT conference.

Chang, C.-W., Huang, G., Lin, B., & Chuah, C.-N. (2015). Leisure: Load-balanced network-wide traffic measurement and monitor placement. *IEEE Transactions on Parallel and Distributed Systems, 26*(4), 1059-1070.

Chen, M.-H., Tien, Y.-C., Huang, Y.-T., Chung, I.-H., & Chou, C.-F. (2016). A low-latency two-tier measurement and control platform for commodity SDN. *IEEE Communications Magazine, 54*(9), 98-104.

Chowdhury, S. R., Bari, M. F., Ahmed, R., & Boutaba, R. (2014). *Payless: A low cost network monitoring framework for software defined networks.* Paper presented at the 2014 IEEE Network Operations and Management Symposium (NOMS).

Cisco. Sampled NetFlow [Cisco IOS Software Releases 12.0 S]. from https://www.cisco.com/c/en/us/td/docs/ios/12_0s/feature/guide/12s_sanf.html

Claise, B. (2004). Cisco systems NetFlow services export version 9.

Clayman, S., Mamatas, L., & Galis, A. (2016). *Efficient management solutions for software-defined infrastructures.* Paper presented at the Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP.

Consortium, O. S. (2014). OpenFlow Switch Specification Version 1.5.0 (Protocol version 0x06): OpenFlow Networking Foundation.

Cormode, G., & Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms, 55*(1), 58-75.

David Patterson, J. H. (2014). Computer Organization and Design MIPS Edition: The Hardware/Software Interface (5th ed., pp. 800). USA: Elsevier.

Dixit, A., Hao, F., Mukherjee, S., Lakshman, T. V., & Kompella, R. (2013). Towards an elastic distributed SDN controller. *SIGCOMM Comput. Commun. Rev., 43*(4), 7-12.

Duffield, N. (2004). Sampling for Passive Internet Measurement: A Review. *JSTOR, 19*(3).

Dusi, M., Bifulco, R., Gringoli, F., & Schneider, F. (2014). *Reactive logic in software-defined networking: Measuring flow-table requirements.* Paper presented at the Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International.

El-Azzab, M., Bedhiaf, I. L., Lemieux, Y., & Cherkaoui, O. (2011). *Slices isolator for a virtualized OpenFlow node.* Paper presented at the Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on.

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

Estan, C., Varghese, G., & Fisk, M. (2003). *Bitmap algorithms for counting active flows on high speed links.* Paper presented at the Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement.

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

Fundation, O. N. (2012). Software-defined networking: The new norm for networks. *ONF White Paper, 2*, 2.6-6.1.

Gangwal, A., Conti, M., & Gaur, M. S. (2017). *Panorama: Real-time bird's eye view of an OpenFlow network.* Paper presented at the Networking, Sensing and Control (ICNSC), 2017 IEEE 14th International Conference on.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems, 29*(7), 1645-1660.

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

Haleplidis, E., Pentikousis, K., Denazis, S., Salim, J. H., Meyer, D., & Koufopavlou, O. (2015). Software-defined networking (SDN): Layers and architecture terminology.

Han, Y., Jeong, T., Yoo, J.-H., & Hong, J. W.-K. (2016). *FLAME: Flow level traffic matrix estimation using poisson shot-noise process for SDN.* Paper presented at the NetSoft Conference and Workshops (NetSoft), 2016 IEEE.

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

Huang, G., Lall, A., Chuah, C.-N., & Xu, J. (2009). *Uncovering global icebergs in distributed monitors.* Paper presented at the Quality of Service, 2009. IWQoS. 17th International Workshop on.

Iannaccone, G., Diot, C., McAuley, D., Moore, A., Pratt, I., & Rizzo, L. (2004). The CoMo white paper. *Intel Research Cambridge, Tech. Rep. IRCTR-04-017, September*.

IDC. (2016). IDC members., from https://www/idc.com/getdoc.jsp?containerId=prUS41005016

Izard, R. (2016). Fast-Failover OpenFlow Groups. from https://floodlight.atlassian.net/wiki/display/floodlightcontroller/How+to+Work+with+Fast-Failover+OpenFlow+Groups

Jarschel, M., Zinner, T., Höhn, T., & Tran-Gia, P. (2013). *On the accuracy of leveraging SDN for passive network measurements.* Paper presented at the Telecommunication Networks and Applications Conference (ATNAC), 2013 Australasian.

Jarschel, M., Zinner, T., Hoßfeld, T., Tran-Gia, P., & Kellerer, W. (2014). Interfaces, attributes, and use cases: A compass for SDN. *IEEE Communications Magazine, 52*(6), 210-217.

Jose, L., Yu, M., & Rexford, J. (2011). *Online Measurement of Large Traffic Aggregates on Commodity Switches.* Paper presented at the Hot-ICE.

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

Khattak, Z. K., Awais, M., & Iqbal, A. (2014). *Performance evaluation of OpenDaylight SDN controller.* Paper presented at the Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on.

Khondoker, R., Zaalouk, A., Marx, R., & Bayarou, K. (2014). *Feature-based comparison and selection of Software Defined Networking (SDN) controllers.* Paper presented at the Computer Applications and Information Systems (WCCAIS), 2014 World Congress on.

Kumar, A., Sung, M., Xu, J. J., & Wang, J. (2004). *Data streaming algorithms for efficient and accurate estimation of flow size distribution.* Paper presented at the ACM SIGMETRICS Performance Evaluation Review.

Kuźniar, M., Perešíni, P., & Kostić, D. (2015). *What you need to know about sdn flow tables.* Paper presented at the International Conference on Passive and Active Network Measurement.

Lee, S., Kim, J., Shin, S., Porras, P., & Yegneswaran, V. (2017). *Athena: A Framework for Scalable Anomaly Detection in Software-Defined Networks.* Paper presented at the Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on.

Lim, C. L., Moffat, A., & Wirth, A. (2014). *Lazy and eager approaches for the set cover problem*. Paper presented at the Proceedings of the Thirty-Seventh Australasian Computer Science Conference - Volume 147, Auckland, New Zealand.

Lin, P., Hart, J., Krishnaswamy, U., Murakami, T., Kobayashi, M., Al-Shabibi, A., . . . Bi, J. (2013). *Seamless interworking of SDN and IP.* Paper presented at the ACM SIGCOMM Computer Communication Review.

Liu, Z., Manousis, A., Vorsanger, G., Sekar, V., & Braverman, V. (2016). *One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon*. Paper presented at the Proceedings of the 2016 ACM SIGCOMM Conference, Florianopolis, Brazil.

Malboubi, M., Wang, L., Chuah, C.-N., & Sharma, P. (2014). *Intelligent sdn based traffic (de) aggregation and measurement paradigm (istamp).* Paper presented at the IEEE INFOCOM 2014-IEEE Conference on Computer Communications.

Matt Mathis, J. M. (1996). Diagnosing Internet Congestion with a Transport Layer Performance Tool. from http://www.isoc.org/inet96/proceedings/d3/d3_2.htm

Mccauley, J. (2014). Pox: A python-based openflow controller.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., . . . Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review, 38*(2), 69-74.

Medved, J., Varga, R., Tkacik, A., & Gray, K. (2014, 19-19 June 2014). *OpenDaylight: Towards a Model-Driven SDN Controller architecture.* Paper presented at the Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014.

Megyesi, P., Botta, A., Aceto, G., Pescapé, A., & Molnár, S. (2017). Challenges and solution for measuring available bandwidth in software defined networks. *Computer Communications, 99*(Supplement C), 48-61.

Megyesi, P., Botta, A., Aceto, G., Pescapè, A., & Molnár, S. (2016). *Available bandwidth measurement in software defined networks.* Paper presented at the Proceedings of the 31st Annual ACM Symposium on Applied Computing.

Micheel, J., Braun, H.-W., & Graham, I. (2001). *Storage and bandwidth requirements for passive Internet header traces.* Paper presented at the Workshop on Network-Related Data Management, in conjunction with ACM SIGMOD/PODS.

Mininet. (2015). Mininet version 2.2.1. http://mininet.org/overview/ (Version 2.2.1). Retrieved from http://mininet.org/overview/

Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., Curtis, A. R., & Banerjee, S. (2010). *DevoFlow: cost-effective flow management for high performance enterprise networks*. Paper presented at the Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Monterey, California.

Mohan, V., Reddy, Y. J., & Kalpana, K. (2011). Active and passive network measurements: a survey. *International Journal of Computer Science and Information Technologies, 2*(4), 1372-1385.

Moshref, M., Yu, M., Govindan, R., & Vahdat, A. (2015). DREAM: dynamic resource allocation for software-defined measurement. *ACM SIGCOMM Computer Communication Review, 44*(4), 419-430.

Naous, J., Erickson, D., Covington, G. A., Appenzeller, G., & McKeown, N. (2008). *Implementing an OpenFlow switch on the NetFPGA platform.* Paper presented at the Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems.

Nascimento, M. R., Rothenberg, C. E., Salvador, M. R., Corrêa, C. N., de Lucena, S. C., & Magalhães, M. F. (2011). *Virtual routers as a service: the routeflow approach leveraging software-defined networks.* Paper presented at the Proceedings of the 6th International Conference on Future Internet Technologies.

Ng, E., Cai, Z., & Cox, A. (2010). Maestro: A system for scalable openflow control. *Rice University, Houston, TX, USA, TSEN Maestro-Techn. Rep, TR10-08*.

OpenvSwitch. (2017). Open vSwitch version 2.5.5. Retrieved 17 Feb, 2017, from http://openvswitch.org/releases/NEWS-2.5.2

Padhi, R., Unnikrishnan, N., Wang, X., & Balakrishnan, S. (2006). A single network adaptive critic (SNAC) architecture for optimal control synthesis for a class of nonlinear systems. *Neural Networks, 19*(10), 1648-1660.

Panchen, S., Phaal, P., & McKee, N. (2001). InMon corporation's sFlow: A method for monitoring traffic in switched and routed networks.

Patel, M., & Pandya, A. (2017). Edge Computing: Design a Framework for Monitoring Performance between Datacenters and Devices of Edge Networks.

Pfaff, B., Lantz, B., & Heller, B. (2009). OpenFlow Switch Specification Version 1.0. 0: December.

Pfaff, B., Lantz, B., & Heller, B. (2012). OpenFlow Switch Specification Version 1.3. 0: OpenFlow Networking Foundation.

Phaal, P., & Lavine, M. (2004). sflow version 5. *Specification. sFlow. org*.

Phaal, P., Panchen, S., & sFlow.org. Packet sampling basics. from http://www.sflow.org/packetSamplingBasics/index.htm

Rasley, J., Stephens, B., Dixon, C., Rozner, E., Felter, W., Agarwal, K., . . . Fonseca, R. (2015). Planck: Millisecond-scale monitoring and control for commodity networks. *ACM SIGCOMM Computer Communication Review, 44*(4), 407-418.

Rebecchi, F., Boite, J., Nardin, P.-A., Bouet, M., & Conan, V. (2017). *Traffic monitoring and DDoS detection using stateful SDN.* Paper presented at the Network Softwarization (NetSoft), 2017 IEEE Conference on.

Robitza, W., Ahmad, A., Kara, P. A., Atzori, L., Martini, M. G., Raake, A., & Sun, L. (2017). Challenges of future multimedia QoE monitoring for internet service providers. *Multimedia Tools and Applications*, 1-24.

Ros, F. J., & Ruiz, P. M. (2014). *Five nines of southbound reliability in software-defined networks.* Paper presented at the Proceedings of the third workshop on Hot topics in software defined networking.

Schmid, S., & Suomela, J. (2013). *Exploiting locality in distributed SDN control.* Paper presented at the Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.

Schweller, R., Gupta, A., Parsons, E., & Chen, Y. (2004). *Reversible sketches for efficient and accurate change detection over network data streams.* Paper presented at the Proceedings of the 4th ACM SIGCOMM conference on Internet measurement.

SDxCentral. (2016). SDNCentral Exclusive: SDN Market Size Expected to Reach $35B by 2018. from https://www.sdxcentral.com/articles/announcements/sdn-market-sizing/2013/04/

Shahmir Shourmasti, K. (2013). *Stochastic switching using openflow.* Institutt for telematikk.

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

Su, Z., Wang, T., & Hamdi, M. (2017). JOTA: Joint optimization for the task assignment of sketch-based measurement. *Computer Communications, 102*, 17-27.

Su, Z., Wang, T., Xia, Y., & Hamdi, M. (2014). *FlowCover: Low-cost flow monitoring scheme in software defined networks.* Paper presented at the Global Communications Conference (GLOBECOM), 2014 IEEE.

Su, Z., Wang, T., Xia, Y., & Hamdi, M. (2015). CeMon: A cost-effective flow monitoring system in software defined networks. *Computer Networks, 92*, 101-115.

Suh, J., Kwon, T. T., Dixon, C., Felter, W., & Carter, J. (2014). *OpenSample: A low-latency, sampling-based measurement platform for commodity SDN.* Paper presented at the Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on.

Sun, P., Yu, M., Freedman, M. J., Rexford, J., & Walker, D. (2015). Hone: Joint host-network traffic management in software-defined networks. *Journal of Network and Systems Management, 23*(2), 374-399.

Sünnen, D. (2011). *Performance evaluation of openFlow switches.* Swiss Federal Institute of Technology Zurich.

Tootoonchian, A., Ghobadi, M., & Ganjali, Y. (2010). *OpenTM: traffic matrix estimator for OpenFlow networks.* Paper presented at the International Conference on Passive and Active Network Measurement.

Tso, F. P., & Pezaros, D. P. (2013). *Baatdaat: Measurement-based flow scheduling for cloud data centers.* Paper presented at the Computers and Communications (ISCC), 2013 IEEE Symposium on.

Van Adrichem, N. L., Doerr, C., & Kuipers, F. A. (2014). *Opennetmon: Network monitoring in openflow software-defined networks.* Paper presented at the 2014 IEEE Network Operations and Management Symposium (NOMS).

White, G., Palade, A., Cabrera, C., & Clarke, S. (2017). *Quantitative evaluation of qos prediction in iot.* Paper presented at the Dependable Systems and Networks Workshop (DSN-W), 2017 47th Annual IEEE/IFIP International Conference on.

Xie, J., Guo, D., Hu, Z., Qu, T., & Lv, P. (2015). Control plane of software defined networks: A survey. *Computer Communications, 67*(Supplement C), 1-10.

Xu, H., Yu, Z., Qian, C., Li, X.-Y., & Liu, Z. (2017). *Minimizing flow statistics collection cost of SDN using wildcard requests.* Paper presented at the INFOCOM 2017-IEEE Conference on Computer Communications, IEEE.

Xu, H., Yu, Z., Qian, C., Li, X. Y., & Huang, Z. L. a. L. (2017). Minimizing Flow Statistics Collection Cost Using Wildcard-Based Requests in SDNs. *IEEE/ACM Transactions on Networking, PP*(99), 1-15.

Yang, Z., & Yeung, K. L. (2017). *An efficient flow monitoring scheme for SDN networks.* Paper presented at the Electrical and Computer Engineering (CCECE), 2017 IEEE 30th Canadian Conference on.

Yi, K., & Ding, Y. H. (2009, 25-26 April 2009). *32-bit RISC CPU Based on MIPS Instruction Fetch Module Design.* Paper presented at the 2009 International Joint Conference on Artificial Intelligence.

Yu, C., Lumezanu, C., Zhang, Y., Singh, V., Jiang, G., & Madhyastha, H. V. (2013). *Flowsense: Monitoring network utilization with zero measurement cost.* Paper presented at the International Conference on Passive and Active Network Measurement.

Yu, M., Jose, L., & Miao, R. (2013). *Software Defined Traffic Measurement with OpenSketch.* Paper presented at the Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13).

Yuan, L., Chuah, C.-N., & Mohapatra, P. (2011). ProgME: towards programmable network measurement. *IEEE/ACM Trans. Netw., 19*(1), 115-128.

Zander, S., Armitage, G., & Branch, P. (2007). A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials, 9*(3), 44-57.

Zar, J. H. (2013). *Biostatistical Analysis* (5th ed.). New Jersey, USA: Pearson.

Zhang, X., Tune, E., Hagmann, R., Jnagal, R., Gokhale, V., & Wilkes, J. (2013). *CPI 2: CPU performance isolation for shared compute clusters.* Paper presented at the Proceedings of the 8th ACM European Conference on Computer Systems.

Zhou, W., Li, L., Luo, M., & Chou, W. (2014). *REST API design patterns for SDN northbound API.* Paper presented at the Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on.

LIST OF PUBLICATION

Journal Papers:

Paper 1:

Tahaei, H., Salleh, R., Khan, S., Izard, R., Choo, K.-K. R., & Anuar, N. B. (2017). A multi-objective software defined network traffic measurement. *Measurement, 95*, 317-327. (ISI Indexed Q1, Impact Factor 2.359)

Paper 2:

Tahaei, H., Salleh, R. B., Ab Razak, M. F., Ko, K., & Anuar, N. B. (2018). Cost Effective Network Flow Measurement for Software Defined Networks: A Distributed Controller Scenario. *IEEE Access, 6*, 5182-5198. (ISI Indexed Q1, Impact Factor 3.244)