# Assessing the Accuracy of Performance Modelling in Software Defined Networks

by

Jordan Ansell

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Engineering
in Network Engineering.

Victoria University of Wellington
2017

# Abstract

Analytical modelling and experimental measurement can are used to evaluate the performance of a network. Models provide insight and measurement provides realism.

For software defined networks (SDN) it is unknown how well the existing queueing models represent the performance of a real SDN network. This leads to uncertainty between what can be predicted and the actual behaviour of a software defined network.

This work investigates the accuracy of software defined network queueing models. This is done through comparing the performance results of analytical models to experimental performance results.

The outcome of this is an understanding of how reliable the existing queueing models are and areas where the queueing models can be improved.

# Acknowledgements

The following people have made this thesis possible, supporting me and the work throughout the past year. I would like to thank them all.

- My Supervisors, Prof. Winston Seah and Dr. Bryan Ng
- My partner Yi Yin, Aunt, family and friends, for their support
- Prof. Ying Dar Lin and Prof. Lai (NCTU, Taiwan) and Prof. Joel Sommers (Colgate University, USA) for their time and guidance
- Peers who have helped and kept me sane – Jarrod, Alex and all those in the labs on AM Level 4
- Siyun, Mark, Patricia and the other administration staff for their dedication to supporting us students when we need it

A big thank you to the RSNZ-JSPS scholarship that has sponsored me throughout the duration of this research.

# Contents

# Chapter 1

# Introduction

Mathematical analysis can be used to inform the management and configuration of a network in order to optimise network performance. Traditional networks have undergone analysis using these methods for understanding performance. However greater variability within the network environment is introduced by software defined networking (SDN), insisting the need to better understand how the network will perform under this paradigm.

Existing analytical work on software defined networks is limited to only a few papers. Such analysis allows the performance of a network to be predicted, with insight into what factors affect the results. In addition to being limited in number, existing analytical queueing work lacks real-world validation of results. This leaves uncertainty as to the reliability of the analytical performance results of SDN networks. Thus the existing analytical work of SDN networks leads to a grey area and the level to which the results represent reality is unknown.

In this thesis the performance results of an SDN network will be measured and compared to the results predicted by the existing queueing models. Without this step, it is impossible to relate the results of analysis to the real world. From this comparison future analytical models are informed of the features of an SDN network that impact the performance.

## 1.1 Motivation

SDN networks behave differently to traditional networks. Compared to traditional networks, there is a lack of understanding of how SDN's differences affect performance.

The existing work suffers from the following issues:

1. Models are validated through discrete-event or numerical simulations only [12, 14, 15, 6].
2. They assume a software defined network will behave the same as a traditional network. (See Section 1.3.3).
3. Rely on the simplification of exponentially distributed service and arrival times. This is an invalid assumption [17].

The most relevant of these to this thesis is issue 1. Simulation can validate a model for the assumptions made and this can lead to *some* insight into potential pitfalls. However without experimental validation the existing models cannot relate to the real world in a truly meaningful way.

These circumstances lead to analytical models whose results may not represent the performance of real SDN networks. Once the accuracy and limitations of a model are known improvements can be made.

## 1.2 Objectives

The aim of this research is to evaluate the gap between SDN analytical models and the real world network performance they represent, and provide insight into ways that the existing models might be improved. This will be done through the following objectives.

1. Quantify the accuracy of the existing SDN queueing models by comparing the analytical and experimental results

2. Highlight features of an SDN network which could improve the results of the analytical models based on literature and experimental observation.

The remainder of this chapter will introduce the topics relevant to this thesis and survey the existing related works.

## 1.3 Background

### 1.3.1 Software Defined Networks

Software defined networking is a recent paradigm in networking which centralises the network control plane and increases the flexibility of managing a network's data plane [13]. It is defined by physically separate packet forwarding and decision making — respectively the data and control planes. The control plane is logically central, in the form of a controller. This provides a single point for network management and data aggregation. The data plane must communicate with the controller before establishing rules for forwarding traffic.

SDN also aims to provide network administrators with a open and programmable interface for controlling the behaviour of the network. Control resides within software and outside of vendors' forwarding hardware.

SDN switches embody the data plane. They carry network traffic, enforce flow rules and communicate with the control plane. They pass flow information from the data plane to the controller and receive instructions back from it.

If not managed correctly placing the controller at the centre of the network can lead to performance degradation. In the worst cases it becomes a bottleneck and single-point of failure for an entire network.

Flows are a way to categorise network traffic passing through a network. The simplest case represents the traffic within a connection between sending and receiving hosts (e.g. a single host connecting to a website).

More coarse flows can be an aggregate of many connections, such as all the hosts connecting to a website. OpenFlow rules are carried out by matching flows to actions.

*OpenFlow* [1] is the most established [13] communication protocol across the interface between the control and data planes of an SDN network. OpenFlow defines the messages which pass between the switches and controller, and the structure of rules for handling traffic in the data plane. OpenFlow version 1.3.5 is the more mature and commonly imlpemented version and will be used in this investigation.

### 1.3.2 Mathematical Analysis

Mathematical analysis is a tool that can be used for investigating the performance of a system. This is done by abstracting components of a system and modelling their behaviour and interactions. Once a model is seen to represent a particular system its parameters can be changed and the average performance and limitations of the system can be explored for different situations. This is without the overhead of implementation. Model parameters can be tuned to see how they affect the overall performance of the system.

Other kinds of analysis include simulating the system and directly measuring the behaviour of a system. These provide greater realism and also some level of insight, while the cost and rigidity increase the closer to the real-world system the analysis method gets.

In order to judge whether an analytical model represents a given system, the results of the model and real-world system must be compared. The mathematical rigour of a model can be proven under particular assumptions, but without comparison to the real-world system some level of uncertainty will always remain.

[1] https://www.opennetworking.org/sdn-resources/openflow

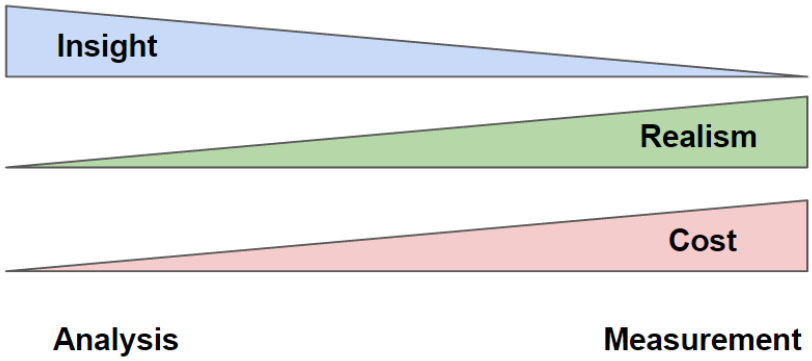



Figure 1.1: Comparing analytical and experimental performance evaluation methods.

#### Queueing Theory

Queueing theory is an established branch of mathematical analysis useful in examining the behaviour of networking systems. In the past it has been used to explore the behaviour of communication networks, assisting with optimising configurations and evaluating performance in order to improve aspects of the network system.

Queueing theory enables stochastic processes to be used, as opposed to methods like Network Calculus which requires deterministic bounds. The output of a queueing model is the average performance of a queueing system. Performance can be expressed in terms of throughput, average queue length or average waiting time in the queue.

### 1.3.3 SDN Requires Performance Analysis

This section describes the particular features of SDN networks that make it different from a traditional network and justifies why these mean analysis is necessary.

SDN networks are:

- Flexible in the logic used to arrive at forwarding decisions.

- Accessible to low cost, commodity hardware implementations.

These points can lead to complications in the data path.

Traditional packet forwarding is well defined and deeply seated in existing hardware. The Internet Engineering Task Force (IETF) and the Institute of Electrical and Electronics Engineers (IEEE) set in place standards for how packets are structured and the protocols for handling them. Vendors of network equipment give life to these standards when creating networking hardware and the software for controlling them.

SDN augments the traditional technologies in the name of flexibility and gives administrators and developers the ability to make their own decisions. The result is a coalition between the switching fabric and CPU. Functionality is not necessarily restricted in the same way as traditional networks, allowing greater freedom. While this could come with compatibility concerns, relevant to this thesis it leads to potential degradation in performance.

From a hardware perspective, the restrictions of standards mean that hardware can be specific and efficient in what it does at the expense of rigidity in the management of the network. The switching fabric used is capable of high processing speeds. By changing the expectations of what the network can do, the existing simple hardware needs assistance from less specialised and slower hardware to execute the new actions. This is particularly the case with SDN, where existing commodity hardware is expected to be coerced into fitting the paradigm. Again, this may lead to variable and reduced performance [5].

Flexibility arises from the *software* part in a software defined network. A network administrator can decide exactly how it wants to treat packets, and have these decisions change in response to behaviour in traffic and network. Doing so creates significant variability in network behaviour. This greater variability implies less certainty in the way that the network will perform and encourages the need for understanding the system's behaviour and what affects the resulting performance.

The separate and centralised controller typically leads to physical distance and a shared resource, which amounts to an increase in delay [18]. Additionally the performance across that distance is subject to usual network uncertainties. A resource between several devices can become a bottleneck and traffic can be lost if it becomes overwhelmed. This leads to delays which can impact the forwarding performance and are unseen in traditional networks.

Due to the open nature of SDN, the software and hardware required to deploy it is available and cheap. Being open to developers, this encourages a great variety of novel functionality. This also makes experimental performance measurement more accessible. By measuring the characteristics of traffic travelling through the network the performance performance of the network can be better understood. To quantify this variable performance through measurement alone can be very costly. More value, in the form of insight, could be gained from analytical models.

While these changes are not objectively negative, they lead to areas where uncertainty in the performance of a network is a problem. Thus, while lacking in the literature, the insight of an analytical approach is beneficial for understanding the performance of software defined networks.

## 1.4 Related Work

There is limited analytical work investigating the performance of software defined networks. Existing work is shared between two branches of mathematical analysis – queueing theory and network calculus. This section summarises the notable existing work.

Work on network calculus is included here for completeness, but the experimental results are not compared to the results of these models in this thesis.

### 1.4.1 Existing Queueing Models

The existing queueing models for SDN performance are described here. They are:

- Simple M/M/1-S feedback queueing model [12].
- Single and double switch models using Jackson network queueing models [14, 15]
- Comparison of controller architectures using queueing models [7]
- QBD analytical model [6].

#### Jarschel *et al.*

Jarschel *et al.* [12] propose a simple M/M/1-S model, the first analytical work on an OpenFlow network with TCP traffic. This considers the case of a single switch and controller (see Figure 1.2), modelling them separately as M/M/1 and M/M/1/K queues respectively. Data plane traffic triggers messages to the controller with probability $P_{nf}$, the probability of no-flow-entry existing for an arriving flow. The paper also describes experimental results of packet service rates for several common OpenFlow 1.0 switches and controllers. The analytical results are validated only through simulation.

Jarschel *et al.* extend this work with a GI/GI/1 model [11], giving generalised results for the simple SDN model. Modelling the switch as a GI/M/1 queue and the controller as M/GI/1, and simplifying assumptions of traffic travelling to the controller. This work shows the level of dependence of an OpenFlow switch's service time on the packet payload size and the flow rules within the flow table. Additionally the dependence of the total network sojourn time on the probability $P_{nf}$ and the controller's service time is shown.

**Mahmood *et al.***

Mahmood *et al.* [14] derive a queueing model to express the same scenario as Jarschel *et al.* [12]. They show how the assumptions of a Jackson network can be used for the data plane and how the $P_{nf}$ value could be more accurately modelled to prevent modelled traffic from revisited the controller. The original work is extended [15] to allow for more switches, grouping together the data plane elements as a Jackson network and placing the controller outside of this. Through simulating the model, the effect of a three switch data plane on sojourn times was shown. The cumulative distribution function (CDF) and probability distribution function (PDF) of the sojourn distribution are also calculated.

**Goto *et al.***

Goto *et al.* use quasi-birth-death (QBD) processes to model a single switch and controller SDN system. Priority scheduling is assumed on the switch for controller's egress traffic. Unlike other models, the switch has a finite capacity buffer. Instead of a simple probability value, a geometrically distributed flow size is used to express the arrival of new flows. The results of this model are validated against numerical simulation only.

**Hu *et al.***

Hu *et al.* [7] show how different control plane architectures can impact the service time of the controller creating new flow rules. It was found that a hierarchical controller architecture scales better than single or other distributed architectures analysed. This evaluates only the computational complexity within the data plane of establishing flows-rules for traffic on the data plane, and not the delay experienced by traffic on the data plane.

The single switch and controller queueing models from Jarschel *et al.*, Mahmood *et al.* and Goto *et al.* are examined on more detail in Chapter 2.

The analytical results of Jarschel *et al.* and Mahmood *et al.* are compared to experimental results in Chapter 6. The analytical results of Goto *et al.* are not included as this model investigates the specific case of priority queueing on the controller egress queue.

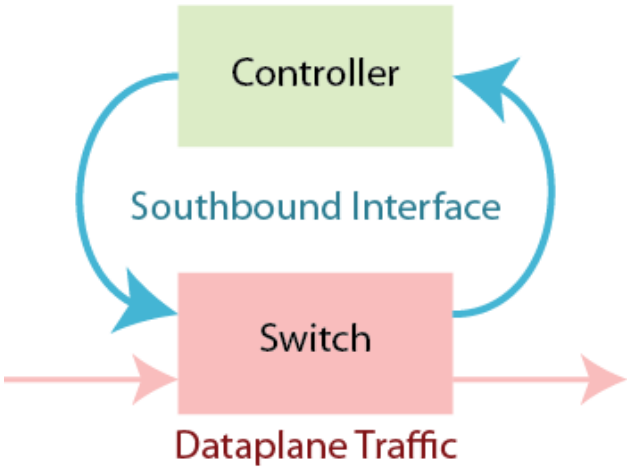



Figure 1.2: The single switch SDN system

### 1.4.2 Network Calculus

Network calculus is used for deterministic network modelling of boundary performance. It's useful for examining worst-case scenarios, as opposed to the expected performance of queueing theory. It has been employed to analyse the performance of an SDN system in several papers.

Using a network calculus approach, Azodolmolky *et al.* [4] model and analyse a hierarchical controller architecture for intra-control plane delay and queue length. They then analyse the same features between data and control planes [3]. Huang *et al.* [9] use a network calculus model for exploring QoS configuration of aggregated flows. Osgouei *et al.* [16] analyse the upper limits of a virtualised SDN environment, handling new flows and considering the additional layer packets must pass through.

### 1.4.3 Limitations of Current Work

The following are factors which are lacking in the work listed above:

1. The existing work assumes:
   (a) Exponential inter-arrival times on the data plane, an assumption which has been shown in the past to be false within packet networks [10, 2].
   (b) Only TCP traffic passing through the data plane. UDP traffic is expected to place a greater burden on a controller.
2. No consideration for delay between the control and data plane, an important factor as shown in [18]. This could result in multiple packets reaching the control plane before a flow is installed.
3. All analysis is validated numerically or through simulation, rather than against experimental results.

## 1.5 Outline of Thesis

### Chapter 2: System Description

A description of the features of an SDN network, and how the existing queueing models express these. This includes a summary of networking concepts which is not covered by the queueing models and areas where assumptions the existing models make may not be accurate.

Results of preliminary investigations and existing literature are used as evidence of some of the behaviour mentioned.

### Chapter 3: Measurement Decisions

This chapter describes the test environment and tools used, evaluating the alternatives and justifying the methods used in subsequent chapters.

### Chapter 4: Service Times

Details the process of measuring the service time parameters of the switch and controller, which are used by queuing models.

### Chapter 5: Network Performance Investigations

The expected behaviour of the network across a range of parameters is explored. The results of this inform the range of parameters used in Chapter 6 to obtain the experimental results. This includes an investigation into anomalous behaviour seen in the switch hardware used.

### Chapter 6: Sojourn Times

Experimental results of the sojourn time and packet loss of an OpenFlow SDN network are measured. The results are presented and discussed.

### Chapter 7: Comparing Existing Models

The results of the analytic SDN queueing models are compared to the experimental results. The differences between these is evaluated and discussed.

### Chapter 8: Conclusion

The results of this research are summarised and discussed.

# Chapter 2

# System Description

This chapter details the SDN system. The scenarios where the existing queueing models are applicable are detailed. The components and features of an SDN network relevant to the traffic and forwarding performance are described. Included is a sample of additional considerations based on literature and experiment which highlight behaviour present in some implementations of OpenFlow-based SDN networks.

It is assumed that the OpenFlow protocol is used to handle the communications between the data and control planes (the *southbound interface*). However much of the discussion will apply to SDN networks general.

## 2.1 Features of Software Defined Network

A traditional network system consists of interconnected switches, each enclosing the forwarding and controller components. Traffic passes through this system and the decision of where to send the traffic next is made on each switch.

Contrasting this, an SDN system consists of separate forwarding and controller components. Traffic moves through and the central controller decides the path through the forwarding components. These components, and how they are modelled will form the basis of the queueing model com-

parison in Section 2.2. This section describes the SDN/OpenFlow system in terms of these components and how they relate to one-another.

There are two paths for packets to take through a switch:

**Data path** pass directly through the data plane of the switch. Processing only occurs in the forwarding component.

**Controller path** travel to the destination through the switch and visit the control plane. Processing occurs at the forwarding component and controller interface component and the controller.

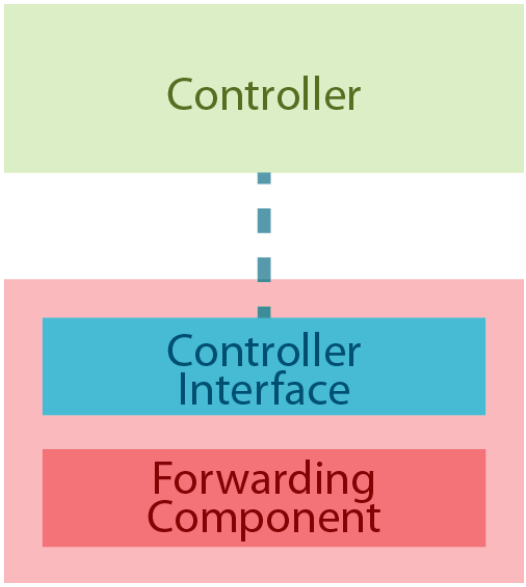



Figure 2.1: Block diagram of SDN components

Switches contain the component for forwarding packets and the component for interfacing and communicating with the controller. The forwarding is performed by matching the headers of an ingress packet against entries in a table of rules called the flow table, then performing the associated list of actions. Multiple tables of rules can exist, and packets move in one direction through these executing actions of the matching rule in

each table. Generally, once processing is complete packets then leave the forwarding component on to a neighbouring switch. This forwarding between switches is called the data path.

The structure of the forwarding element is dependent on each device's implementation. Traffic generally arrives on separate ports [12] and is then processed and matched to actions. Once processed the traffic again departs on separate ports.

The switch component that communicates with the controller also modifies the rules in the flow table following the controller's instruction. To keep behaviour consistent and avoid delays a flow table should be *unused* while it's being modified, which may delay the forwarding component for the duration of the change [5]. Flow setup time is studied in existing work [8, 5]. The controller interface component is responsible for handling the packets which pass to/from the controller, lifting them up from the forwarding component (as with OpenFlow $packet\_in$ message), either encapsulating them in a message or saving them, then sending an OpenFlow message to the controller requesting how to handle the new flow. Similarly moving messages back from the controller to the forwarding element (OpenFlow $packet\_out$ message), and executing instructions not expressible by the hardware alone are handled by this component. This communication with the controller is the controller path.

The controller runs applications which decide how to process and respond to messages from the switch. These applications are also responsible for deciding how the network is monitored, polling the switches for traffic information. As this component of the network is run within a host, it is typically slower than the forwarding component of the switch.

SDN networks deal with the same kind of traffic seen in traditional networks. The complexity of SDN arises in the way switches handle the traffic and communicate with the controller. Traffic in the network is made up of any combination of networking protocols. Each of these has its own flow and packet arrival rate characteristics, sizes and distributions. [10]

Probability distributions provide a way of representing the traffic in a network environment. The arrival of packets is often assumed to be a Poisson process, the inter-arrival times being exponentially distributed. However this has been shown to be an invaild assumption in computer networks [17, 10]. A network environment such as a wide-area network (WAN) will have traffic patterns better fitting of a Pareto distribution.

As stated by Phemius *et al.* [18] and Turull *et al.* [19], the delay of the southbound interface and $flow\_mod$ becomes a significant factor when more than one packet per flow is sent to the controller. This is the case with UDP flows or TCP flow which are mid-flow. However, this is not currently considered in the existing models.

### 2.1.1 Hardware and Software Switches

SDN switches can be implemented either in hardware, software or a hybrid of the two [5]. A software switch will perform all of its functions within the RAM and CPU of the host device. A hardware switch will perform some of the functions (e.g. matching and forwarding) in the hardware (generally TCAM) and call on the CPU when the hardware is unable to accomplish a certain task.

As shown by Jarschel*et al.* the service time of a hardware switch can be orders of magnitude faster than a software switch. However, if this speed comes from using TCAM memory, the time to modify rules can be on the order of seconds [5].

While not considered in the existing SDN queueing models, and outside the scope of this thesis, some flow rules instruct for more actions with a greater number of steps than just forwarding – for example modifying VLAN tags in packet headers. More steps in hardware or software incur greater delay.

### 2.1.2 Network Scenarios

The manner in which a switch and controller interact can be classified as either reactive or proactive. Various combinations of characteristics exist in real world deployments. These terms are described here.

#### Reactive Network

In a reactive network the switch only receives rules in response to changes in the data-plane traffic. This can be in the form of flows which enter the switch being forwarded to the controller when they do not match any existing flow rules. Alternatively these reactive changes can be caused by monitoring existing static flows, and adjusting the paths or traffic engineering settings.

In the simplest case, a purely reactive OpenFlow switch will start with only the instruction to consult the controller when a new flow is unmatched. Unmatched flow rules trigger a $packet\_in$ event in OpenFlow, the controller then decides how to handle the unmatched flow. Thus new flows of packets result in the controller instructing the switch to install new flow rules

Depending on the granularity of flows, the frequency of flow table misses will change. When granularity is fine (e.g. based on ports) there will be more controller path traffic than with coarse granularity (e.g. based on a range of IP addresses). The proportion of new flow rules or probability of there not being an existing flow rule has been considered in the existing models at the finest of granularity, but can be used at any level of granularity. Jarschel *et al.* [12] and Mahmood *et al.* [14] call this $P_{nf}$.

An example of a reactive network in practise is a network where each connection must be considered before installing a rule, based on network policy or calculating the best path for packets to take. Limited flow rule table size in a switch is another situation where this might be favourable.

### TCP and UDP Traffic

As per the work of Phemius *et al.* [18] and Turull *et al.* [19], under the assumption that each new source-destination pair creates a new flow, the transport layer protocols TCP and UDP can behave very differently in a SDN network.

TCP must wait for a reply before sending data traffic. In a reactive SDN network the first packet of a TCP flow will trigger a new flow rule to be installed in the switch before the data is sent.

In general, UDP does not wait for a response and sends data packets immediately. In a reactive SDN network the flow rule is not installed before the data packets of a new connection arrive. This results in much greater load on the network [18], being a combination of additional traffic reaching the controller and potentially of additional repeated instructions being sent to the switch.

Assuming TCP traffic simplifies the network traffic model significantly but avoids a serious concern regarding the reactive network scenario. As noted in existing work [18], if a TCP connection's flow rule is removed mid-session a similar problem to the UDP issue will also arise.

The data plane of the existing OpenFlow standards rely on the controller to change behaviour in response to events similar to UDP. There is not currently support for the switch to limit the traffic sent to the controller in ways that could avoid this.

### Proactive Network

In a proactive network scenario the new flow rules are triggered by events on the controller rather than resulting from events in the network. The controller can modify the rules over time independent of the behaviour of the data-plane. In general, this will lead to more predictable traffic passing through the southbound connection. Data plane traffic will not be congesting the connection. An example of this is routing or traffic engi-

neering decisions based on time of day, changes in network policy or a network administrator making decisions manually.

In its pure form no packets need to be passed between the data path and the control plane. When there is no interaction between the traffic of the data and control planes, the delay of data path is not influenced by the delay on the control plane – assuming there is no data path delay caused when modifying flow table entries. An exclusively proactive network scenario is not necessarily something which benefits from modelling of both the control and data plane traffic together.

### 2.1.3 Observations of Preliminary Work

Before executing the main body of work with switch hardware, experiments were performed using ZodiacFX hardware to explore the behaviour of the SDN system. This summary represents the discoveries made during that work, relevant to building a better understanding of the general SDN system.

#### Multiple Switch Service Times

The existing queueing models assume that the switch has a single service time whether packets are forwarded along the data plane or result in a $packet\_in$ event being sent to the controller. This short experimental work shows that this is not the case, and that there are distinct service times depending on the path the message is travelling.

There are three possible paths for a packet arriving at the switch.

1. Traffic arriving from the data plane which is directly forwarded out on the data plane (data-data).
2. Traffic arriving from the data plane which is triggers a $packet\_in$ event to be forwarded to the controller (data-controller).
3. Traffic arriving from controller as a $packet\_out$ event, which is then forwarded out on the data plane (controller-data).

The preliminary results in Figure 2.2 show that each of these three paths result in different times and distributions.

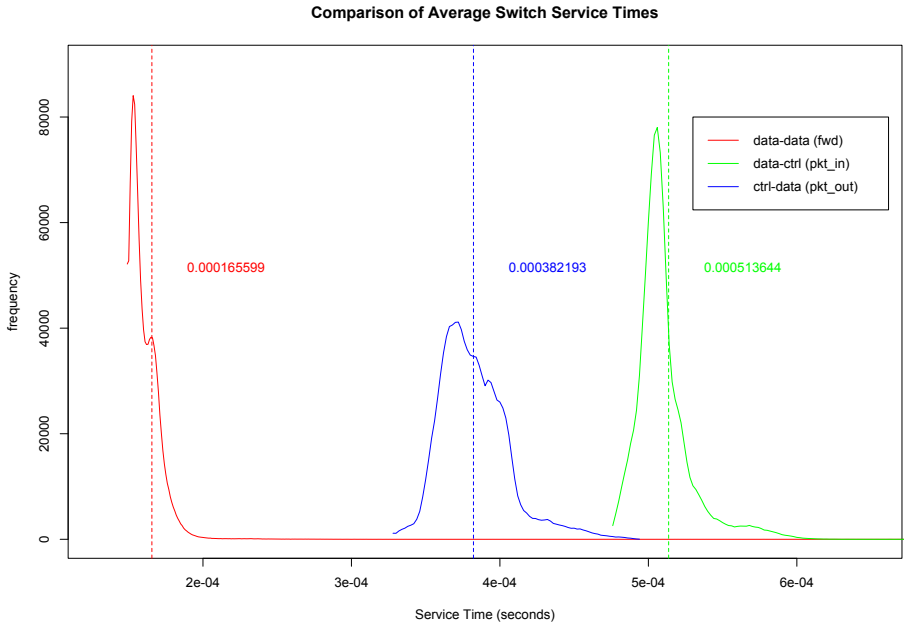



Figure 2.2: Mean switch service times of ZodiacFX, showing each distribution of delay in each path

Considering the forwarding and controller interface components of a switch and what is needed to fulfil these roles, this makes sense [8]. A switch needs to move packet data between ports according to existing rules and communicate with the controller for modifying those rules. Packets arriving and being forwarded along the data plane require only headers being matched against existing flow rules. Communication with the controller requires a network session and network stack management (e.g. TCP in an OpenFlow session). Passing through the network stack creates a lot of overhead, something the data plane can avoid. The difference between these service times is more significant for hardware switches than for software or hybrid switches [5].

When performing measurements for the purposes of doing mathematical analysis, it's important to take these differences into account. A simple way to do this, though not necessarily ideal, is by adjusting controller's modelled service time by the difference between it and switch's data plane service time. Depending on the switch implementation this difference in delay can allude to a third queueing process occurring in the switch between the data-plane and the controller, as one could expect from a hardware switch passing messages up to the CPU. A more complicated way of modelling this could be to using different classes of jobs with varied service times in order to model this with a single switch queue.

## 2.2 Features of Existing Queueing Models

This section describes the existing single-switch SDN queueing models by breaking them down to the features they express and mathematical assumptions they make.

The existing models are:

- **Jarschel *et al.*'s Model** M/M/1-S feedback queueing model [12].
- **Mahmood *et al.*'s Model** Single switch model using Jackson network queueing models [14]
- **Goto*et al.*'s Model** QBD analytical model [6].

Throughout this thesis the name of each paper's primary author will be used to refer to each of these models. Unless stated these models assume the First-In-First-Out queueing discipline.

### 2.2.1 Jarschel's Model

**Traffic Behaviour**

- (A) Inter-arrival times of packets are assumed to be exponential.

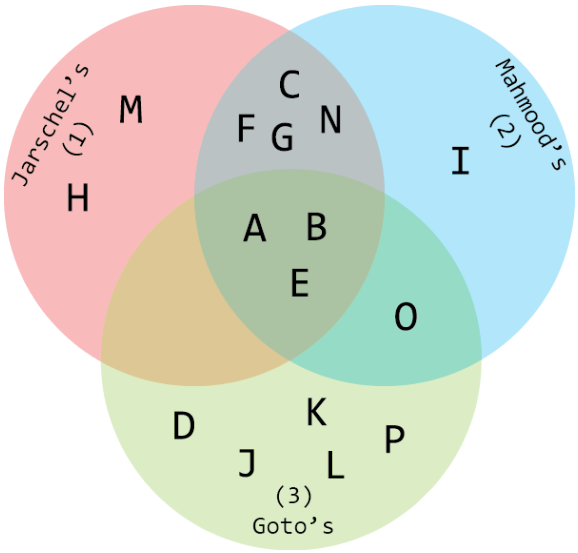



Figure 2.3: Venn diagram showing the relationship between the existing SDN queueing models

- (B) It is assumed that only TCP traffic is present in the network.
- (C) New flows are represented by $P_{nf}$ at the switch deciding to send packets to the controller.

**Switch Behaviour**

- (E) Purely reactive flow rules.
- (F) Uses $M/M/1/\infty$, modelling the time between ingress traffic events and service times as the time between events in a Poisson process.
- (G) Queue is shared between ingress data traffic and traffic from the controller.
- (H) $P_{nf}$ is the proportion of traffic leaving the switch that is sent to the controller. Mathematically the model *does not distinguish* between ingress data traffic and traffic from the controller when applying this proportion.

**Controller Behaviour**

- (M) Uses M/M/1/S, modelling the time between ingress traffic events and service times as the time between events in a Poisson process. The S-capacity queue enables packet loss to be expressed.
- (N) Egress traffic is sent to the switch.

### 2.2.2 Mahmood's Model

**Traffic Behaviour**

- (A) Inter-arrival times of packets are assumed to be exponential.
- (B) It is assumed that only TCP traffic is present in the network.
- (C) New flows are represented by $P_{nf}$ at the switch deciding to send packets to the controller.

**Switch Behaviour**

- (E) Purely reactive flow rules.
- (F) Uses M/M/1/$\infty$, modelling the time between ingress traffic events and service times as the time between events in a Poisson process.
- (G) Queue is shared between ingress data traffic and traffic from the controller.
- (I) $P_{nf}$ is the proportion of traffic leaving the switch that is sent to the controller. Mathematically the model *distinguishes* between ingress data traffic and traffic from the controller when applying this proportion.

**Controller Behaviour**

- (O) Uses M/M/1/$\infty$, modelling the time between ingress traffic events and service times as the time between events in a Poisson process.
- (N) Egress traffic is sent to the switch.

### 2.2.3 Goto's Model

**Traffic Behaviour**

- (A) Inter-arrival times of packets are assumed to be exponential.
- (B) It is assumed that only TCP traffic is present in the network.
- (D) Flow size is modelled using a geometrically distributed value. Arrival of new flows is based of this value.

**Switch Behaviour**

- (E) Purely reactive flow rules.
- (J) Uses M/M/1/K, modelling the time between ingress traffic events and service times as the time between events in a Poisson process. A K-capacity queue, enabling packet loss to be modelled.
- (K) Two queues, one for priority traffic from the controller, and the other for ingress data traffic. Priority queue does not pre-empt queue server.
- (L) Parameter for flow size determines arrival of new flows.

**Controller Behaviour**

- (O) Uses M/M/1/$\infty$, modelling the time between ingress traffic events and service times as the time between events in a Poisson process.
- (P) Egress traffic is sent to the switch's priority queue.

### 2.2.4 Differences Between Models

As seen in Figure 2.3 the models are similar in many ways. As they represent the same system this is expected. Below are the features of the models which differentiate each from the other existing models.

The following numbers will be used to reference the models that express the listed feature. 1) Jarschel's Model; 2) Mahmood's Model; 3) Goto's Model.

**Flows:** Model 1 and 2 represent new flows with $P_{nf}$. Whereas model 3 models the arrival of a new flow as a geometric distribution.

**Switch:** Model 3 models a priority queue with finite buffer. Packets from the controller are given priority. Model 1 and 2 assume traffic from the controller and ingress data traffic enter in the same queue. Model 2 and 3 prevent traffic from visiting the controller twice.

**Controller:** Only Model 1 models a finite capacity buffer at the controller.

# Chapter 3

# Measurement Decisions

This chapter describes the test environment and tools used in later chapters to perform the experiments. The alternative options are described and how well they fit the requirements is evaluated. The performance of the selected tools and environments are also evaluated. The methods used to measure network performance in subsequent chapters are justified here.

## 3.1 Experiment Requirements

The following list of requirements refers to the needs of the equipment used to perform the experiments in chapters 4, 5 and 6. They cover the hardware used to generate packets, to send packets and to record packet data. [1]

1. **Identify**: Identify each packet uniquely when sent and received.
2. **ProbDist**: Inter-packet time is variable following a probability distribution (for example, the Exponential distribution).
3. **MaxRate**: Send packets up to the link's capacity.

[1] The numbers and bold titles in this list will be used to refer to each requirement in following sections.

4. **Reliable**: Produces reliable measurements (accurate, precise, low overhead).
5. **Obtainable**: Hardware is already obtainable or of low cost.
6. **PrepTime**: Does not require excessive time to prepare.
7. **Format**: Data recorded in an accessible format.

Note that the overhead of requirement 4 refers to the effect the measuring equipment has on the measurements made, such as additional delay of a host's network stack and network interface card when sending or receiving Ethernet frames.

## 3.2 Packet Generator

The following tools were considered when selecting a traffic generator. This is divided into hardware and a software categories. The hardware category contains specialised devices which create and release packets directly into the network cables. The software category is for programs that run on a computer's operating system, and generate packets via the hardware available in the host machine.

### 3.2.1 Hardware Tools

In general, the results obtained from a proprietary hardware traffic generator will have accurate measurement results. The drawbacks are generally a high cost and lack of flexibility.

#### XenaCompact

Layer 2/3 traffic generator. Scripting language available for orchestrating multiple flows with various rates, packet sizes and custom packet header

XenaCompact: xenanetworks.com/test-chassis/xenacompact-chassis-solution

and payload information. Only supports a constant packet rate per flow. A XenaCompact is available for use.

#### STG-10G

Stateful traffic generator. Combines D-ITG and DPDK libraries, so can generate at packet rates defined by various probability distributions with a high degree of low level control. D-ITG and DPDK are described later in this chapter. This device is very expensive.

This is included as example of the best possible option.

#### NetFPGA

Programmable network interface card. Would require existing code or time to learn to program, but could create the traffic inter-arrival and labelling characteristics required. Accuracy of results is dependent on programming. NetFPGA hardware is available for use.

### Software Tools

In general software tools are more flexible but are subject to performance limitations/overheads of the operating system (OS) and hardware within which they run. All the tools listed here are available online as open-source software or are free with open APIs.

#### Nping

User-level software using the OS network stack. Command line tool for generating arbitrary packets and sending them via raw sockets. Simple and quick to use. Suffers an upper bound of reliability at 1000pps (from operating system).

STG-10G: www.ecdata.com/stg-10g.html

### Scapy

User-level software using the OS network stack. Tool for generating arbitrary packets from within a Python environment. Access to the full range of Python capabilities. Suffers an upper bound of reliability at 1000pps (from operating system).

### D-ITG

User-level software using the OS network stack. This tool has many pre-programmed probability distributions for setting the inter-arrival and packet size attributes of a flow. Many flows with scripted start times can be run at once.

Through trials it was found to be unreliable when unsupervised, frequently crashing, making automation a big problem. Tools for processing the data files included, down to the flow level of detail.

### Data Plane Development Kit (DPDK)

Kernel-level software which replaces the network stack providing direct access to the network interface. PktGen is a python interface for generating traffic with DPDK. Missing features can be added, like probability distributions and extracting packet information. API requires time to learn.

### PF_Ring

Kernel-level software which replaces the network stack providing direct access to the network interface. Comes with usable codes examples for sending and receiving packets (respectively zsend and zcount). Missing features can be added, like probability distributions and extracting packet information. Has trial access to advanced network driver features for improved accuracy.

| | Identify | ProbDist | MaxRate | Obtainable | PrepTime |
|---|---|---|---|---|---|
| **XenaCompact** | X | | X | X | X |
| **STG-10G** | X | X | X | | X |
| **NetFPGA** | E | E | X | X | |
| **Nping** | | | | X | X |
| **Scapy** | X | E | | X | X |
| **D-ITG** | X | X | | X | X |
| **DPDK** | E | E | X | X | |
| **PF_Ring** | E | E | X | X | X |

X:Passes requirement (blank):Fails requirement E:extra work required

Table 3.1: Comparison of Traffic Generators

### 3.2.2 Comparison of Traffic Generators

Missing from this table are Experiment Requirements 4 and 7 as these are requirements of capturing the traffic measurements.

The three hardware options are eliminated here despite their promises of more reliable results. The XenaCompact, while accessible is not able to vary the traffic's probability distribution. The effort required to program a NetFGA and the cost of a STG-10G also eliminate them.

Even with the flexibility offered by D-ITG with regard to varying traffic parameters it suffers unreliability from the overhead of the OS it runs on, making it unsuitable for the resolution required to measure a devices forwarding delay. DPDK with its power suffers the cost of time required to learn and implement using the libraries.

One of the tools is able to meet all the requirements, but only with extra work to program the needed features. This is PF_Ring. While similar in mission to DPDK time required to learn API is reduced by the existing example programs within PF_Ring. The features missing from the example code that need to be implemented are the ability to save packet information and sending with inter-arrival times determined by a probability distribution.

## 3.3 Delay Measurement Tools

There are two important components to capturing results. One is the act of capturing the packet data. The second is keeping time between the sent and received packets. Packet data consists of a packet ID and the times sent and received. The tools and approaches here contribute to the Experiment Requirements 4 and 7.

### 3.3.1 Data Capture Tools

In terms of capturing the packet and time information the following tools were considered.

#### TCPDump

User-level software using the OS network stack. Records packet information from the network as a PCAP file. As it uses system time, the resolution of the time captured is unreliable.

#### PF_Ring

When used it receives the packet information already, which can be saved if programmed to extract and record it. Uses an internal clock which is be accurate to below 100 nanoseconds.

#### DAG Card

A network card built for the purpose of accurately capturing network traffic data. Records to ERF format, which can be converted post-capture to more universal PCAP if required. Hardware would need to be purchased.

DAG Card: https://www.endace.com/compare-dag-card-models.html

| | Reliable | Format | Obtainable | PrepTime |
|---|---|---|---|---|
| **TCPDump** | | X | X | X |
| **PF_Ring** | X | E | X | |
| **DAG Card** | X | X | | X |

X:Passes requirement (blank):Fails requirement E:extra work required

Table 3.2: Comparison of Data Capture Tools

### 3.3.2 Comparison of Delay Measurement Tools

From this comparison PF_Ring is the clear choice. Again though, it will require modification to save packet data.

The method used to record time suffers a small overhead, even while not using the OS network stack. The time taken to pass the packet up to the user program leaves room for random fluctuations in the host system to influence time measurements.

#### Hardware Timestamps

The NIC hardware used features the ability to record the time packets enter or leave. As this feature can avoid the overhead of the measuring host's OS and kernel, tools which can use it are able to meet the reliable requirement.

Through experimentation it was found that the timestamp of received packets was configurable in the hardware, resulting in a timestamp within the raw packet data accessible to all raw capture methods. It was also found that accessing the timestamp of sent packets was less successful. The send time must be retrieved by the NIC driver, a task not possible with PF_Ring and the hardware used. To access the received hardware timestamps in PF_Ring additional modifications are necessary.

Having the facility of hardware timestamps would be the most idea scenario for a host-based time measurement. There is always a limit to the accuracy. One must accept that what is being used and progress.

### 3.3.3 Clock Synchronisation

In order to make time measurements across multiple hosts the clocks on these host must be kept synchronised. Existing networking protocols exist, including Network Time Protocol (NTP) and Precision Time Protocol.

Existing tools for synchronising clocks are:

- systemd-timesync (using NTP)
- Chrony (using NTP)
- IEEE 1588 Precision-time-protocol (Linux PTP) [1]
- Dedicated hardware (atop PTP)

The best option here is the precision time protocol. It uses hardware timestamps to estimate the delay between hosts and provide sub-microsecond synchronisation.

## 3.4 Evaluating Performance of Tools Selected

As modifications were made to PF_Ring to enable some additional features, these features need to be evaluated to check they work. The extra features are:

- Inter-arrival time of packets can be set to use an exponentially distributed random variable.
- Packet information is saved efficiently and in a format that's easy to read later.

The act of using the performance results will determine how good the packet information format is. The efficiency of saving information will reflect in the ability to capture times correctly.

So to evaluate these changes the following results are collected:

- Difference between send and receive inter-packet times.
- Received distribution is exponential.

### 3.4.1 Measured Error in Inter-packet Times

Using a constant inter-packet time on the traffic generator, the inter-packet time of the sent and received packets is measured. These are compared to what they should be, and the difference is the error measured. While it's important the traffic generating host can send packets as expected, the receiving host must also be able to receive them to show that there is no interference in the path. The receiver's results are included here, as the senders results show only what is expected.

Figure 3.1 shows the expected and observed trends.

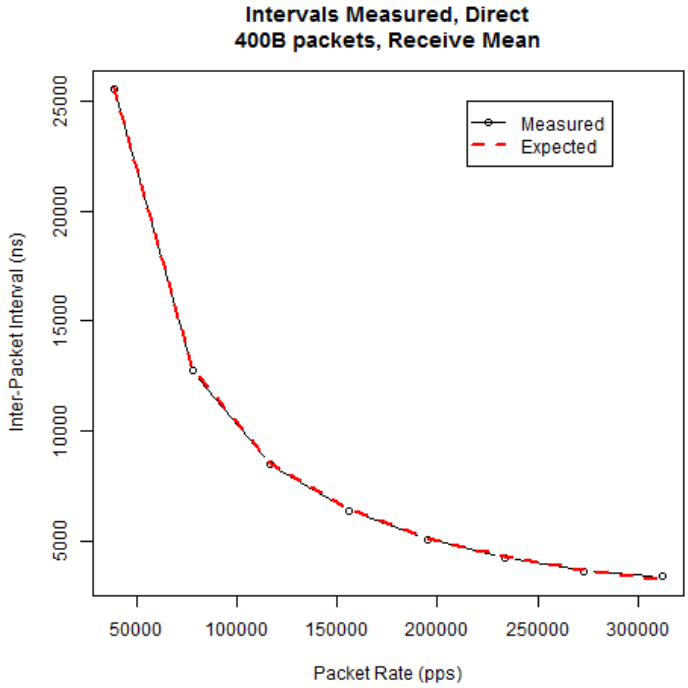



Figure 3.1: Inter-packet times measured on receiving host compared to inter-packet delay programmed on sending host

The observed trend appears to be exactly what is expected – as the packet rate increases the time between packets should be the reciprocal of the packet rate. To quantify how well the observed trend matches, the

difference between these is calculated and graphed in Figure 3.2. These results show that the error in the mean is below 1.1% for all bit rates except 312000 pps – not shown here because it is -0.05. At 0.125Gbps the error increases to 1.3% and at 1Gbps the error is 5%.

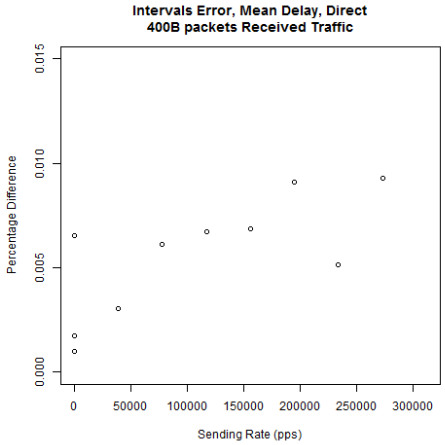
Median send error is much higher, measured to be below 20%. The packets are received at intervals equal to the smallest difference with the PF_Ring method (256 nano seconds). This could indicate that many packets are arriving in batch.

This effect can be explained by looking at Figure 3.2 (c). The red line indicated the mean, and the green line indicated the median. The mean of the data is placed between two clusters of data. The balance of these places the mean at the expected value, but in between the two clusters.

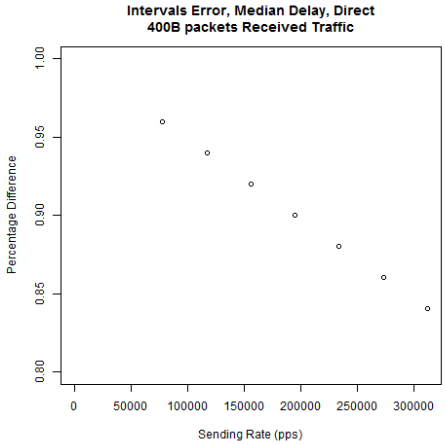
This shows that the traffic generator and measurements used can reach a mean inter-arrival delay close to that desired. However this is split into two modes, which are balanced around the mean. This is the cost of the flexibility of using a general purpose host machine to vary packets according to a probability distribution.

However, this does not prevent measurements of constant inter-packet delays being used, and as explained in the next section, the output required for the measurements of a probability distribution are still valid.

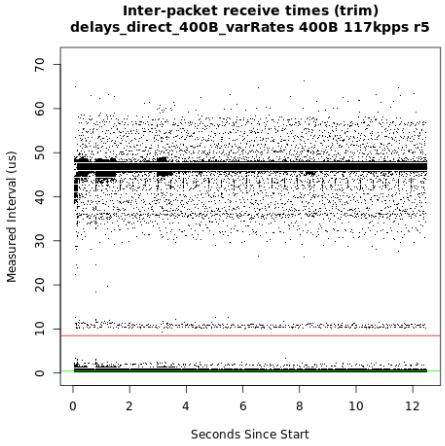
This could be improved with feedback from the hardware on the sender, however this would have required significant modifications the drivers and further modifications to the software used – a task well beyond the scope of this thesis.




(a) Receive Interval Mean




(b) Receive Interval Median




(c) Distribution of inter-packet times

Figure 3.2: Inter-packet times on receiving host

### 3.4.2 Exponential Distribution

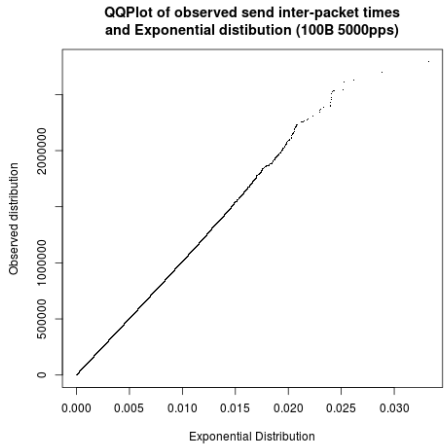
This section presents how well the traffic output by the tool built follows an exponential distribution. Results of one million packets sent with the tool are presented in Figure 3.3 and Figure 3.4. For the kind of plot drawn a straight line indicates the distribution of the measured data matches that of the test data. In this case the test data is exponentially distributed, placed along the x-axis in these figures.

The values chosen for testing are mean packet rates of 5000 pps and 30000 pps, and constant packet size of 100 bytes and 1400 bytes. These represent the high and low end of each parameter. The high end is reaching the limits of what the link can handle.

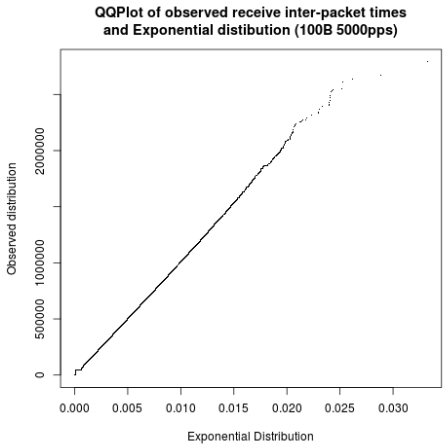
The results shown here are positive. There is a clear linear relationship between the measured and testing data, a very strong indication that the measured data is exponentially distributed. The results for the low end of the test range (5000 pps and 100 byte packets) are perfect. The 30000 pps rate traffic suffers in the low end of the received measurements, seen in the square shape of both packet sizes.

This indicates that in the extreme end the overhead of the host is interfering with the inter-packet distribution. This produces the minimally skewed inter-packet times seen. This is related to the effect seen in Figure 3.2 (c), whereby the limits of the hardware are reached.

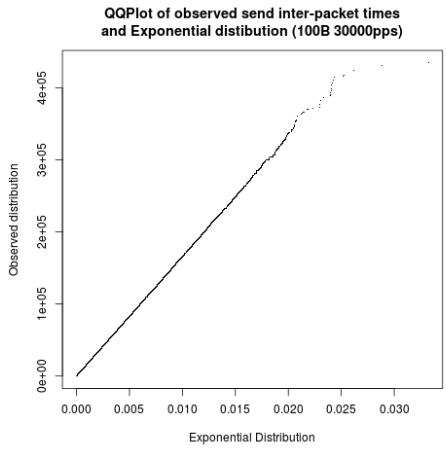
The mean inter-packet time measured on both sender and receiver match for all trials. So while the distribution became skewed in the high end the mean of the data was still expressed correctly.



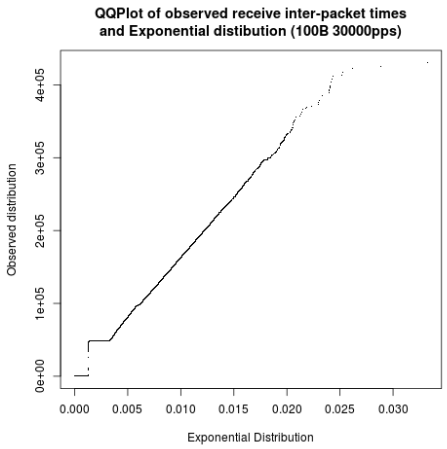

(a) Sent 5000pps




(b) Received 5000pps

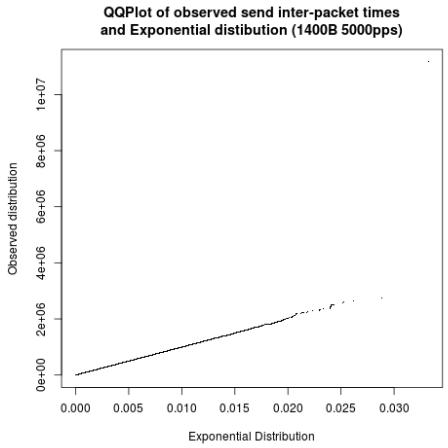



(c) Sent 30000pps

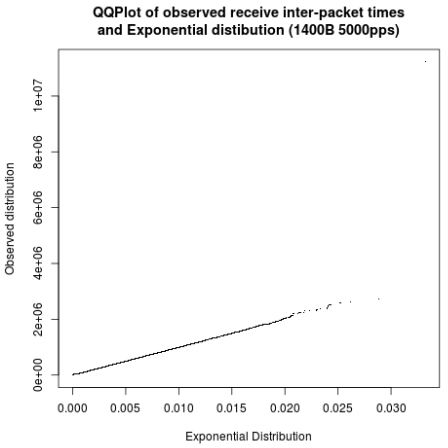



(d) Received 30000pps

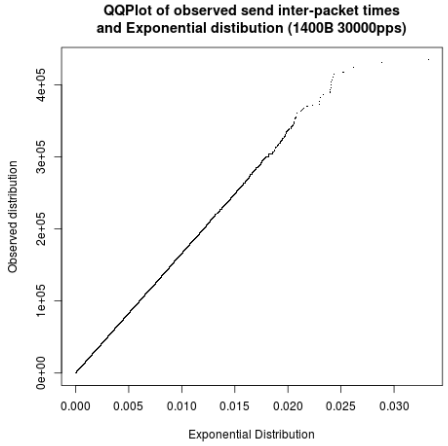
Figure 3.3: Test tool distribution, 100 byte packets



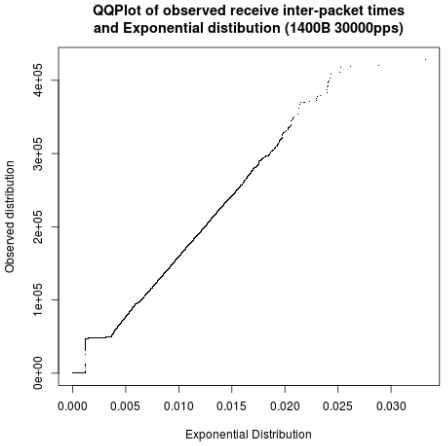

(a) Sent 5000pps




(b) Received 5000pps




(c) Sent 30000pps




(d) Received 30000pps

Figure 3.4: Test tool distribution, 1400 byte packets

## 3.5 Experiment Network Setup

This section outlines the network used to perform the experiments in Chapters 4, 5 and 6.

PF_Ring is selected to send traffic and capture the packet information. PCAP files are generated that can carry a distribution of packet sizes or characteristics. PF_Ring is then used to replay the PCAP file with custom inter-packet times – in the case of the measurements in Chapter 6 this inter-packet time is exponentially distributed.

Messages that could be transmitted (ARP, DHCP, DNS etc) are also prevented from entering the measurement environment.

Hardware timestamps, as described in Section 3.3.2, are used where possible to remove the host-processing delay and obtain more accurate measurements of the delay of the network device.

When using two hosts to measure the delay of packets, clock synchronisation between the two hosts is achieved using IEEE 1588 (PTP)

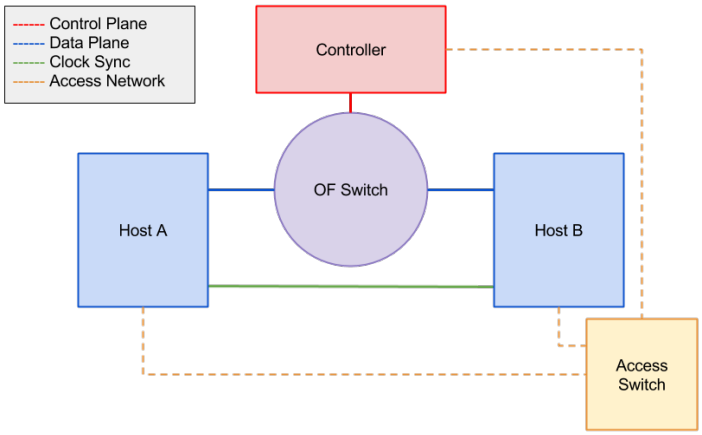



Figure 3.5: Diagram of the testbed

The measurement equipment consists of:

- An Allied Telesis AT-510x SDN switch.
- Two Dell Optiplex 7040 hosts – CPU: i7-4790, RAM: 8GB – additional Intel I350 NIC (2-Port card).
- An Intel NUC as the controller.

#### SDN Configuration

For familiarity reasons the controller application used to manage the switch is Ryu. The switch rules for measuring the service time match on the switch ports, and for later experiments the transport layer port number.

In a reactive network there are two classes of packet in a TCP flow – the first packet, which is used by the controller to determine the flow rule installed. And the subsequent packets, which only pass through the switch. To emulate a TCP flow, unidirectional UDP packets are used. Port numbers are used to differentiate between the first and the subsequent packets. Packets emulating the first flow will be sent with a probability of $P_{\text{nf}}$. Packets emulating the subsequent packets with be sent with a probability of $1 - P_{\text{nf}}$.

A mix of controller and data path traffic are only required when both paths are being measured at the same time (such as in the sojourn experiments in Chapter 6. When only one path is measured, the flows are matched on switch ports and not packet content.

## Methods for Performing Measurements

All of the measurements made follow one of the following two methods. The will be referred to here and in later sections as the *End-to-end Delay Method* and *Network Events Method*.

### End-to-end Delay Method

Two hosts have their clocks synchronised. The times of packets leaving the sending host and arriving at the receiving host are recorded by each respective host. The difference between these times is then calculated as the delay between the hosts.

The difference between the delay when the hosts are directly connected and delay where there is a network device in between can be compared to calculate the delay across just the network device. This removes the processing delay of the source and sink hosts from the device delay measurements.

### Network Events Method

Only one host's clock is used. Packets are intercepted in the network and copies of the packet are sent to the timekeeping host. The difference between the arrival time of the network events is used to calculate the time taken across sections of the traffic's path in the network.

The biggest issue here is the time between sending a packet and its clone.

Using hardware timestamps, measurements of this delay were taken, and the delay is consistently equal to 922 ns. The hardware timestamps have a resolution of 8 ns, meaning this delay is between 918 and 926 ns. The only other measured delay was shown in 1.3% of the results, is exactly 1000 ns.

The delay calculated from these results is a simple difference. The difference between original and cloned packet are reliably consistent. The times reported by the arrival timestamp of network events at the sink can be used to calculate the difference between two packets. Thus the delay of a network device traversed between two consecutive network events will be equal to the difference in time between the arrival of the cloned packets at the sink.

# Chapter 4

# Service Times Measurements

This chapter describes the process of obtaining the packet service parameters required to use the existing analytical models. The service rate represents the processing rate of the hardware in the queueing system, aligning the models' predicted performance results with the real world.

Other parameters describe the traffic, network scenario and policies the system inherits from the controller application. These are assumed to be independent of the service rates. They determine decisions in the modelling phase as behavioural features the model tries to express.

As described in Section 2.1.3, there are several different service times in the SDN/OpenFlow system. These are as follows:

1. **Switch:** Receiving a data packet on the data plane, which then leaves the switch on the data plane
2. **Switch:** Receiving a data packet on the data plane, which then goes to the controller as a $packet\_in$ message
3. **Switch:** Receiving a $packet\_out$ message from the controller, which then sends a data packet out the switch on the data plane
4. **Controller:** Receiving a $packet\_in$ message and respond with $packet\_out$

In existing SDN modelling work, only service times and are considered: the *controller's service rate* and the *data-to-data plane service rate of the switch*. This is the case for the models evaluated in Chapter 7.

## 4.1 Component of Network Delay

Each device, along a packet's path from origin to destination, experiences transmission, processing, and queueing delays. Propagation delay is experienced between each device as the signal passes through the network cable. The *processing delay* of the network devices is the value which is intended to be measured. The *queueing delay* is what queueing models express when calculating average performance.

Propagation times along a few meters length of Ethernet cable below are about five nanoseconds, thus negligible. Queueing delay can be removed by having only a single packet in the system at a time during measurements. Transmission time, the time it takes for the message to be sent by the network hardware, is the only time which is not trivial to remove, and will be included depending on the point where packet times are recorded. In order to eliminate the effect of queueing delay packets are sent at a constant rate, with enough inter-packet delay to avoid packets queueing in the hardware.

## 4.2 Controller Path Service Times

The controller path service times consist of the three times:

1. Controller's service time
2. Time for data packet to be sent as a $packet\_in$ message
3. Time for $packet\_out$ message to be sent as a data packet

All three of these service times can be measured in a single experiment. How to achieve this is detailed below, and shown in Figure 4.1.

### 4.2.1 Experiment Details

These measurements use the Network Events method to measure the delay between packets, with two hosts connected to the switch's data plane The switch is used to clone packets to the sink at each stage of the process.

Traffic is sent in one direction between the source and sink hosts. Upon arriving at the switch, packets are cloned – one is forwarded on to the sink, the other re-enters the switch to be processed and sent to the controller as a $packet\_in$. The link to the controller also passes through the switch, where clones of each packet are sent to the sink. Messages in both directions along this link are thus captured. Once finished at the controller and returned to the switch, the packet is finally received at the sink.

The sink records the times of network events, and enough information about the packet is captured to uniquely identify it and the event it represents. This information includes the MAC address and transport layer port number.

Under the assertion that only one packet should be in the system at a time the packet rate of 10pps is selected as experiments showed this was slow enough to avoid overlap in consecutive receive and send times when passing through the control plane.

Upon receiving a $packet\_in$ message, the controller application is programmed to install a new rule on the switch which matches the destination mac address and send forward to the switch's port belonging to the sink host. For obtaining the service times the priority of this rule was lower than the rule to send a $packet\_in$ event to the controller enabling all packets to be forwarded to the controller. The rule being installed is never matched and as it is always equivalent to a previously installed rule this is overwritten each time a $flow_mod$ message arrives. This ensures all the same behaviour occurs as would happen outside of these tests.

These are averaged results from three trials of 30000 observations, filtered to 98.5% of results to remove extreme outliers. At 10 pps a trial of each packet size takes an hour, including data processing overhead. Three

trials are taken for each packet size, due to time constraints.

First the distribution of service times is examined, then using the information from that the average service time is presented. Following this, a short investigation to understand an unexpected trend is described.

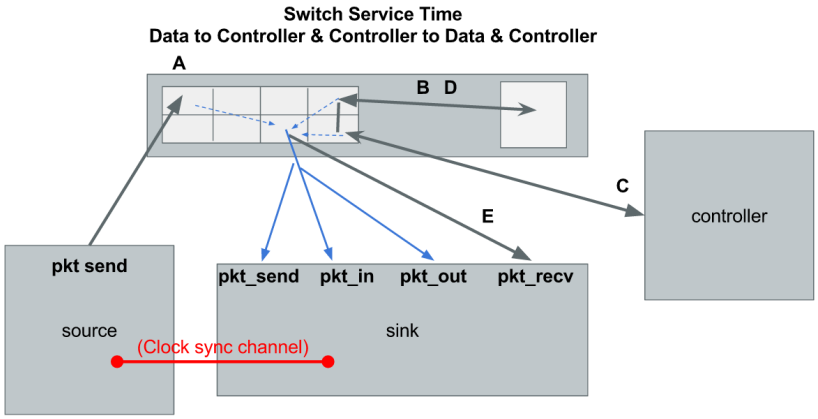



Figure 4.1: Measuring the service time of the OpenFlow switch with messages and controller

### 4.2.2 The Delay Distributions

Figure 4.3 presents histograms of the measured service times. These describe the distribution of the delays measured for each of the desired service times. All the service times show a regular, multi-modal distribution, complimented with clusters of data at different points for each different service time. The regular modes appear at roughly four millisecond intervals, beginning at four milliseconds for Figure 4.3 (a) and (b), and beginning at eight milliseconds for (c). Separating these clusters are areas of very few observed measurements.

The most interesting of these is Figure 4.3 (c), the controller's measured service time. Figure 4.2 shows the service time of the controller measured at the controller using OS-bound software TCPDump. This describes a

trimodal distribution of the controller's service time. When measured in the network at the sink, the same experiments show the multi-modal distribution common among all the service times on the controller path. But the trimodal trend measured on the controller is separate along the x-axis from this common pattern.

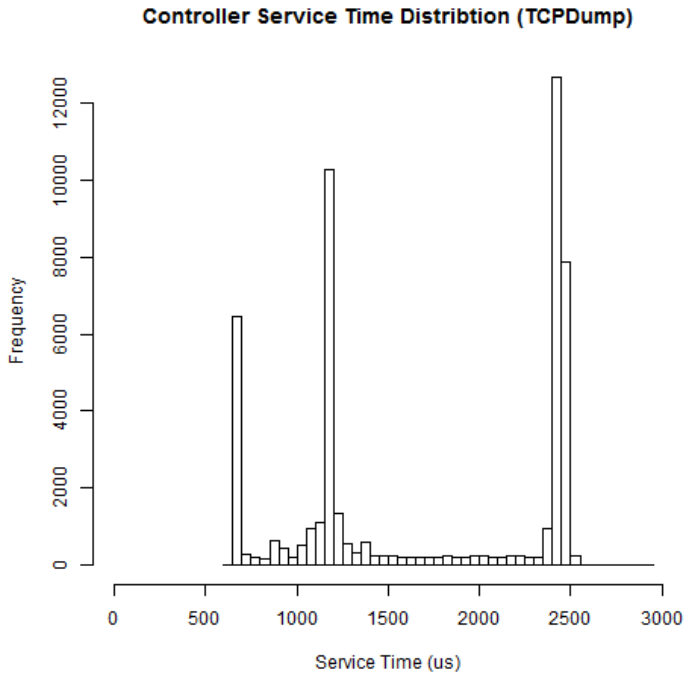
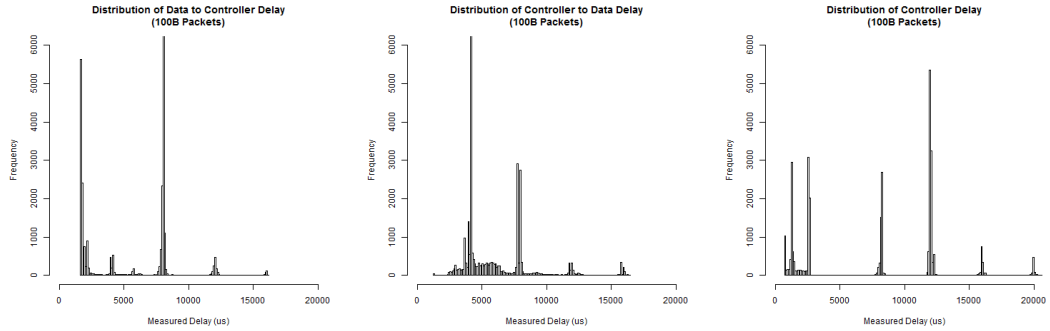



Figure 4.2: Controller service time, measured using TCPDump on the controller

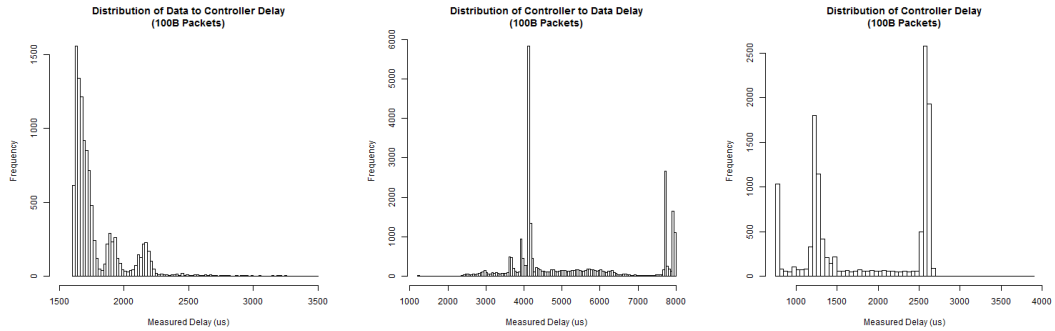
On closer inspection, the separation of two patterns in the service time distributions seems to be present in graph (a) also. Figure 4.4 shows the same delay distributions, trimmed to only those below the first four millisecond interval in (a) and (c), and showing only the first cluster of (b).

In Figure 4.4 (c), the resulting trimodal distribution exactly matches that measured on the controller in Figure 4.2.




(a) Data to Controller (b) Controller to Data (c) Controller

Figure 4.3: Distribution of Controller Path Delays




(a) Data to Controller (b) Controller to Data (c) Controller

Figure 4.4: Distribution of Controller Path Delays (Trimmed)

### 4.2.3 Average Service Times

Based on the service time distributions, the average of both the complete distributions and the trimmed distributions are presented in Figure 4.5.

The overall trend is a decrease in service time as the packet size increases.

| Packet Size (Bytes) | Measured Delay (microseconds) | | | |
|---|---|---|---|---|
| | Data Plane to Controller | Controller | Controller to Data Plane | Total |
| **100** | 5660 | 7570 | 6020 | 19300 |
| **200** | 5720 | 7540 | 6010 | 19300 |
| **400** | 5680 | 7490 | 6000 | 19200 |
| **600** | 5560 | 7450 | 5890 | 18900 |
| **800** | 5640 | 7490 | 5910 | 19000 |
| **1000** | 5610 | 7450 | 5910 | 19000 |
| **1200** | 5610 | 7460 | 5870 | 18900 |
| **1400** | 5610 | 7470 | 5810 | 18900 |

Table 4.1: Mean Delay Measured Over Controller Path (3 SF)

### 4.2.4 Discussion of Results

The trend seen in the measured results is not intuitive. It is usually expected that as the packet size increases the delay across a piece of networking equipment will increase. The average result measured here shows the opposite trend.

By comparing the opposing trends in the complete and trimmed columns of Figure 4.5 and considering the clustering in Section 4.2.2 comparing the same sets of data, it is clear that the data removed by trimming is influencing the average of the data to give this unexpected result.

The average delay measured when a packet is returning to the data plane from the controller still follows the unexpected decreasing trend (see Figure 4.4 (b)). This is because the interval pattern begins within the resulting trimmed data and it cannot be separated from this in a simple manner.

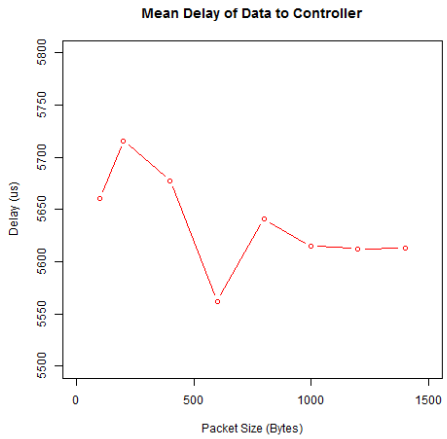
The common pattern of clustering at four millisecond intervals is explained in Chapter 5. This is caused by the unfortunate behaviour of the switch when the forwarding component interacts with the control plane's

interface. The similarity between Figure 4.2 and Figure 4.4 (c) suggest that when this it is trimmed away, delay close to the actual processing delays of the equipment can be seen.

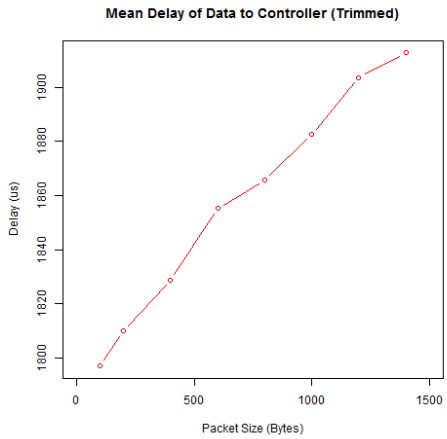
As the experiments in Chapter 6 will observe the same delays as seen in the complete data, the average of the complete data will be used in the queueing models in Chapter 7 to allow a better pairing with the model and the observed sojourn time.

There is a noticeable anomaly in the results for the packet size 600 bytes. This is the result of only using three repetitions. Further repetitions would avoid this.

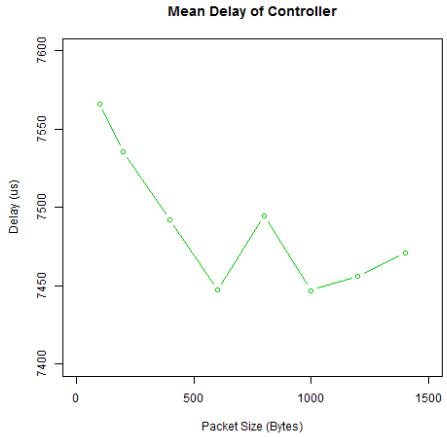
The controller path service times are higher and more inconsistent than expected due to behaviour in the switch.



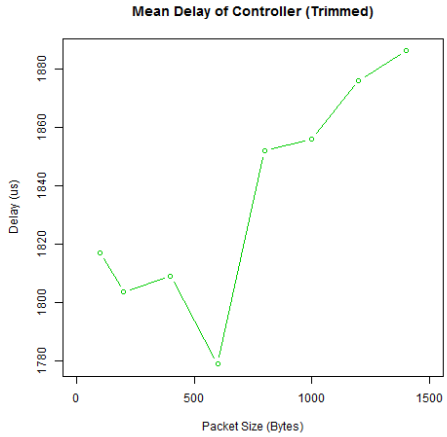

(a) Data to Controller (Mean)



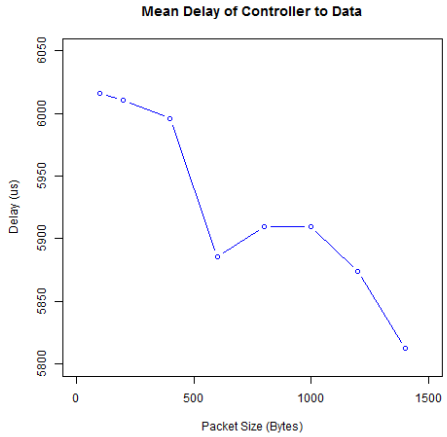

(b) Data to Controller (Trimmed)



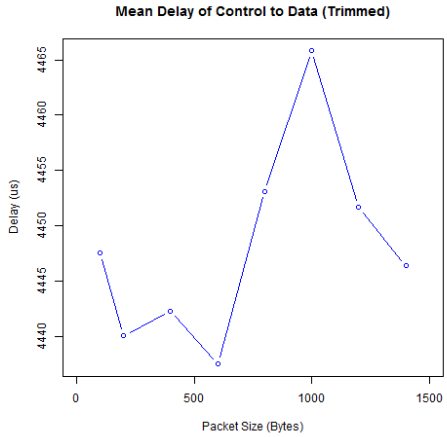

(c) Controller (Mean)




(d) Controller (Trimmed)




(e) Controller to Data (Mean)




(f) Controller to Data (Trimmed)

Figure 4.5: Average of Controller Path Delays – Mean and Trimmed Mean

## 4.3 Switch Service Time

The switch (or *data path*) service time consists of only the delay over the switch when forwarding a packet from one port on the forwarding component to another port on the forwarding component. In general, this is faster than the controller path service times, and faster rates of packets can be sent when measuring.

### 4.3.1 Experimental Setup

This is measured in a more novel manner than the controller paths' service times. The method used is an End-to-end method. The delay measurement contains the result of a varying number of visits to the switch. This allows time for the to-be-forwarded packet to be measured accurately, despite the overhead of the hosts making the measurements. How this equipment is arranged and the packet paths are described in Figure 4.6.

The source host sends packets to the switch, one at a time, recording the sending time. The packet passes through the switch N times, before exiting and arriving at the sink where the receive time is recorded. The clocks of source and sink are synchronised via a dedicated and direct link using PTP. This experiment is repeated for several values of N.

As OpenFlow rules are stateless, there is no way to count the number of times a packet has passed through the switch. Instead, the switch is set up with several cables looping back into itself. OpenFlow rules are statically set so a packet arriving on port A will always leave on port B. Each arrival and departure at the switch is a service of the packet. For each N a different set of static rules are programmed to control the number of times a packet will pass through the switch.

The average delay of each number of loops is calculated. The average difference in time between the loops will be equal to the average service time of the switch. The first second of data is discarded to allow the system to start.

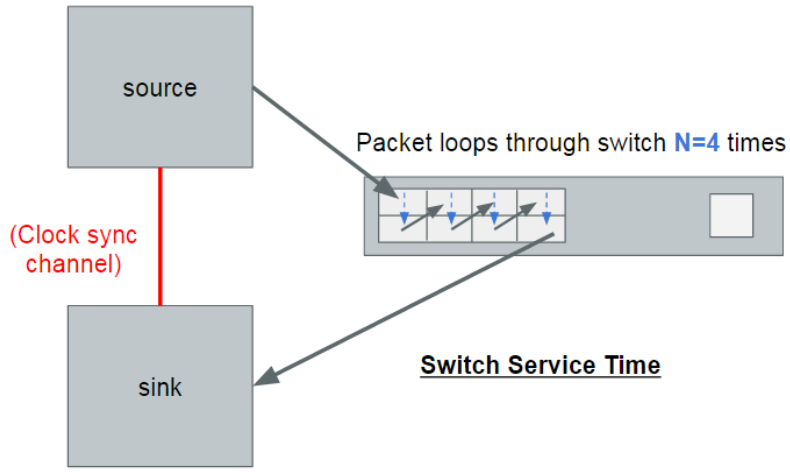



Figure 4.6: Experiment to loop multiple samples of the switch service

To combine the results of several iterations, the average of each repetition is calculated, then the mean over the repetitions is calculated. Each sample is filtered to 99.999% of results recorded results to remove outliers.

### 4.3.2 The Delay Distribution

The distribution of data in each sample is shown in Figure 4.7. A similar distribution is seen for all samples after removing the extreme outliers. The multi-modal shape gives a clear most common value in each sample, and two other modes are spread below this.

### 4.3.3 Average Service Time

The average standard deviations of the average mean, median and mode taken of each iteration, are given in Table 4.2.

The first loop has a higher standard deviation than the other loops. As seen in Figure 4.8 the difference between one and two loops is greater than the difference between the other loops.

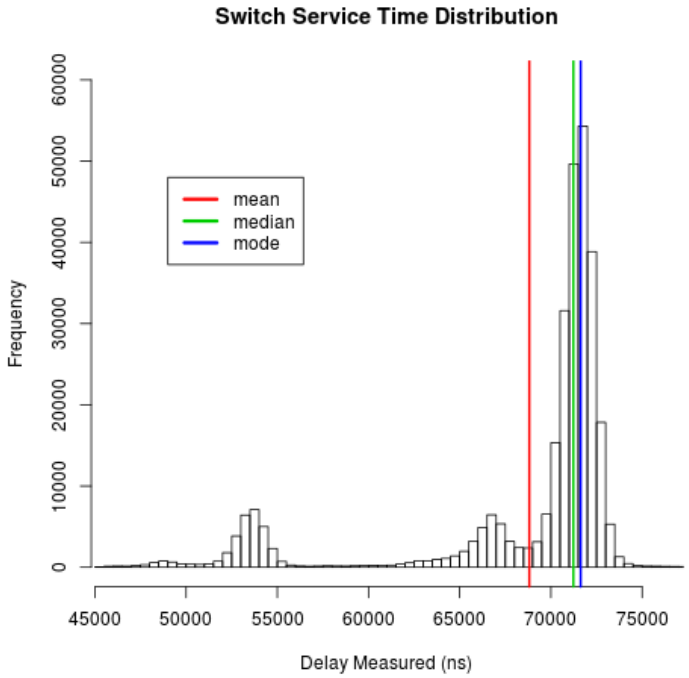


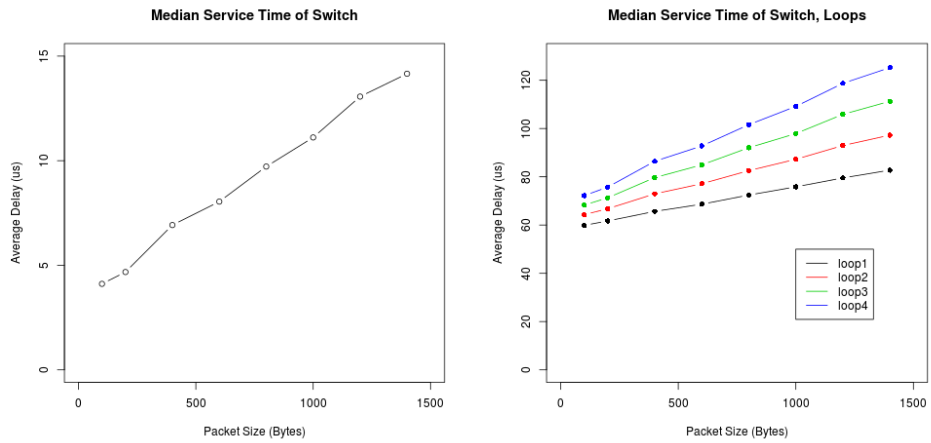

Figure 4.7: Example of the switch service time distribution

The mean is an unstable average to take for representing the results of each sample, shown with the high delay. In terms of the distribution is sits slightly lower than the median and mode.

| | **Mean Delay (nanoseconds)** | **Median Delay (nanoseconds)** | **Mode Delay (nanoseconds)** |
|---|---|---|---|
| **All Loops** | 560.72 | 74.87 | 101.27 |
| **First Loop Removed** | 536.24 | 73.95 | 57.18 |

Table 4.2: Mean standard deviation of repetitions

In order to both minimise the variance of the data used, while also including the average of all four loops, the median is selected as the average service time of the switch.




(a) Switch service times (b) Delay of each loop

Figure 4.8: Average switch service time results

| **Packet Size (Bytes)** | **100** | **200** | **400** | **600** | **800** | **1000** | **1200** | **1400** |
|---|---|---|---|---|---|---|---|---|
| **Measured Delay (us)** | 4.11 | 4.68 | 6.92 | 8.04 | 9.72 | 11.1 | 13.2 | 14.3 |

Table 4.3: Median Delay Measured Over Switch (3 SF)

| **Packet Size (Bytes)** | **Loop 1** | **Loop 2** | **Loop 3** | **Loop 4** | **Total SD** |
|---|---|---|---|---|---|
| **100** | 71.95981 | 30.71218 | 68.57621 | 95.03383 | 140.9098 |
| **200** | 55.09667 | 28.19289 | 160.1195 | 58.50946 | 181.3618 |
| **400** | 82.59962 | 89.3891 | 42.56738 | 45.83121 | 136.8415 |
| **600** | 122.5997 | 90.37067 | 51.80734 | 40.06126 | 165.7904 |
| **800** | 104.8531 | 53.01061 | 75.08234 | 31.28213 | 142.8994 |
| **1000** | 40.73937 | 43.71805 | 79.09748 | 85.91333 | 131.1811 |
| **1200** | 107.7056 | 44.17713 | 114.2414 | 55.70714 | 172.3557 |
| **1400** | 35.73689 | 103.5028 | 91.25435 | 196.526 | 242.7752 |

Table 4.4: Standard deviation of average median service time (ns)

# Chapter 5

# Network Performance Investigations

Before running the sojourn and packet loss experiments, more information about the limits and characteristics of the network needs to be known. Unlike the previous measurements, for the sojourn measurements the traffic in the network cannot be limited to a single packet at a time. The equipment used has a maximum capacity of 1Gbps, which is the equivalent of $1250000 \cdot 100$ byte packets every second.

The measurements in this chapter assess the effect of adding queueing delay and mixing the control and data plane. The service times in Chapter 4 treats the controller and data plane paths separately.

Thus the aim is to:

- Investigate how the network is expected to behave under different packet arrival rates.
- Check there is no state-dependence in the switch.
- Decide what range of parameters will be used when measuring average network delay.

Given the high latency of sending a message to the controller with the

AT x510 switch, it's expected there will be packet loss across the control plane at low traffic rates, relative to the rates handled by the switch path.

## 5.1 Controller Path Parameter Space

This section investigates the behaviour of traffic travelling along the controller path. For this the switch contains a rule to forward all traffic to the controller via a $packet\_in$ message. The controller then replies to the switch with $flow\_mod$ and $packet\_out$ messages. Traffic is sent along this path at a constant packet rate.

The first task is finding the maximum rate that traffic can pass through the controller path. Based on the service time measurements, the average controller path rate of processing is 52 pps. Comparing rates of input traffic with output traffic, 70 pps was the lowest rate that would be allow input to equal output rates.

The time that each packet takes, when being sent at 70 pps is graphed in Figure 5.1. This result indicates there are two points where a queue is encountered along the controller path. As the controller path traffic passes through the switch twice, this is the likely cause.

This is measured with the end-to-end method.

### Delay Interval

Using a network event method the time between consecutive packets is measured on arrival at the switch. Figure 5.2 shows the distribution of consecutive packet times. These results are processed to show the distance from the send rate. For example, the time zero means that the duration since the last packet arrived matches the send rate, and the time -20000 ns means that the packet was early by 20000 ns when compared to the send rate.

The distribution of inter-packet times of $packet\_out$ messages before

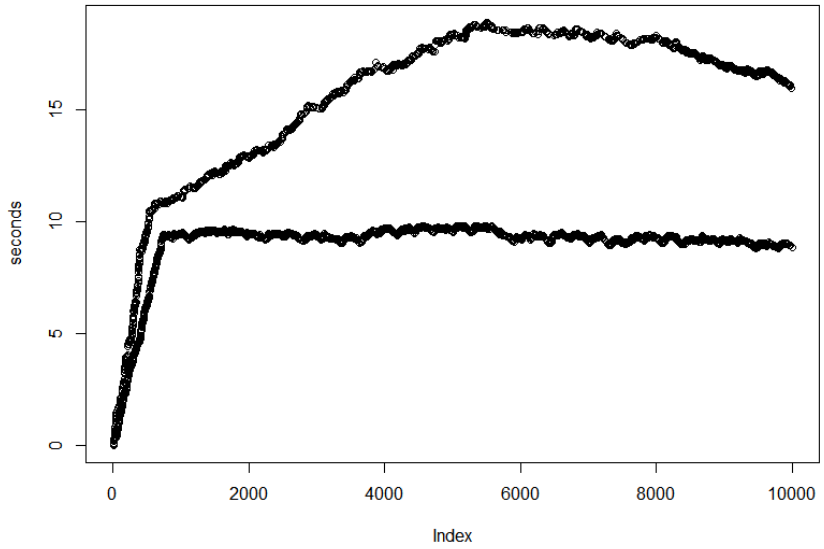


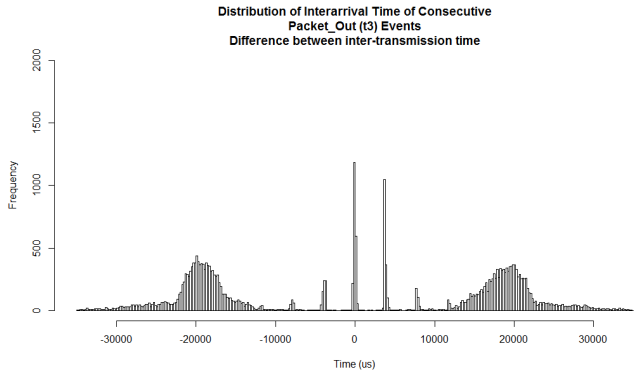

Figure 5.1: Delay of packets, in order sent, when packet rate is 70 pps across the controller path.

passing through the switch (Figure 5.2 (a)) is being coerced into intervals of four milliseconds before they are sent out on the data plane as data packets.

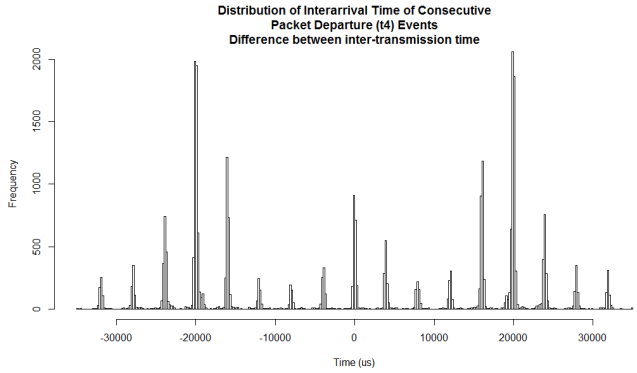
This shows there to be some delay occurring on the controller path, which stalls packets for intervals of four milliseconds. It appears to be happening on the switch when packets move between the forwarding component and the controller interface component.

## Controller Path Summary

To avoid packet loss and see matching send and receive rates of the end hosts, a rate of at most 70pps can be used across the controller path. Higher than this and packets incur significant delay, from both queueing and an unusual 4 ms interval phenomena, caused by something occurring in the switch when moving packets between the control and data planes.




(a) Time between $packet\_out$ events




(b) Time between data packets arriving at sink

Figure 5.2

Relating this to the literature, Bifulco *et al.* [5] indicate issues in the low level implementation of OpenFlow can cause delay like this when writing rules to hardware. The evidence gathered says this is an issue with interacting between the forwarding component and the controller interface in the switch. These points may certainly be related.

## 5.2 Data Path Parameter Space

This section details a short investigation that checks the switch delay at different packet rates, and how the network can be expected to behave during the sojourn experiments. During portions of the tests the data rate will be high enough to stress both the measuring hosts and network devices. This is performed using the end-to-end method so the overhead of the source and sink hosts needs to be checked too.

### 5.2.1 Send Traffic at Maximum Data Rate

Packets at a range of sizes are to be sent the maximum possible bitrate and the resulting delay and packet loss results will be observed. When comparing the rate of 1Gbps to the service time measurements in Chapter 4, it is clear the switch should experience some queueing delay when the packet arrival rate is high.

In this end-to-end method, the *via-switch* connection delay is subtracted from the *direct* source to sink connection delay.

#### Results

Measurements direct from source to sink suffer no packet loss. Measurements over the switch suffer loss during the first few thousand packets. This loss is due to the traffic reaching the maximum service rate at the switch. Delay also increases up until a constant delay "flat-line" is reached.

Because of the packet loss during the start-up period, the first two seconds of each sample will be removed to avoid this effect impacting on the sojourn measurements. Two seconds was decided upon through some trials, where this was enough time to encapsulate the early packet loss.

Interestingly, as shown in Table 5.1, the maximum recorded rate is much lower than the maximum capacity of the link. This indicates the

true maximum rate that the hardware and network are able to send packets at.

| **Packet Size (bytes)** | **100** | **200** | **400** | **600** | **800** | **1000** | **1200** | **1400** |
|---|---|---|---|---|---|---|---|---|
| **1Gbps (pps)** | 1250000 | 625000 | 312500 | 208333 | 156250 | 125000 | 104166 | 89285 |
| **Measured Rate (pps)** | 236967 | 218341 | 143885 | 123457 | 102564 | 89286 | 77519 | 70922 |

Table 5.1: Measured processing rate compared to maximum capacity of Ethernet link

### 5.2.2 Send Traffic at Various Data Rates

The packet rate values used covers two ranges. For a spread of values, the first range is multiples of eighths of 1Gbps (125, 250, 375, 500, 625, 750, 875, 1000 Mbps). The seconds is much smaller, covering 1 Mbps and multiples of 10 below (10, 100 Kbps, and 1 Mbps)

To reduce the number of experiments only one packet size is used. 400 byte packets are used because it shares delay and packet loss characteristics with both the larger and the smaller packet sizes.

#### Results

Through exploring the parameter space an interesting observation is made when inter-packet times are below 1 Mbps. The delay of the direct connection was greater than the delay measured across the switch. This is seen in Figure 5.3 (a) and the data points are removed in Figure 5.3 (b). This is found to be a repeatable observation. Unfortunately this makes the end-to-end method measurements unusable in this range.

There are three sections visible in Figure 5.3 (a). The different sections are grouped by inter-packet times less than 125 Mbps, 125 to 375 Mbps and 500 Mbps and above. The first section is where the delay of the direct connection is greater than the delay of the network. The second section shows a small, premature, increase of delay, similar in quality to that of the

third section. The third section completes the expected increase in delay as the arrival rate of packets gets close to the forwarding rate of the switch and measuring hosts.

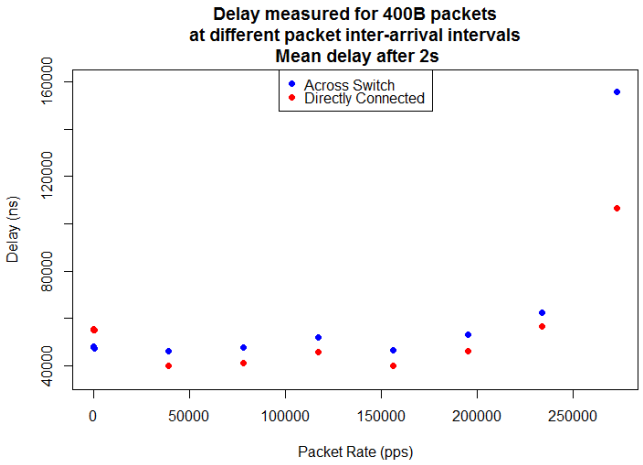
This indicates the delay measured is dependent on the rate of traffic passing through measurement hosts. While this appears to be a problem, at the trialled inter-packet times greater than 125Mbps this trend is consistent across both direct and via-switch measurements. At the high end, the difference increases, indicating the switch is becoming saturated and queueing delay is increasing significantly.

Figure 5.3 (a) shows the separate direct and via-switch delays and Figure 5.3 (b) shows the difference between these, for 125 to 875 Mbps. Note, for 400B packets these data rates translate to 400 to 280 Kpps.

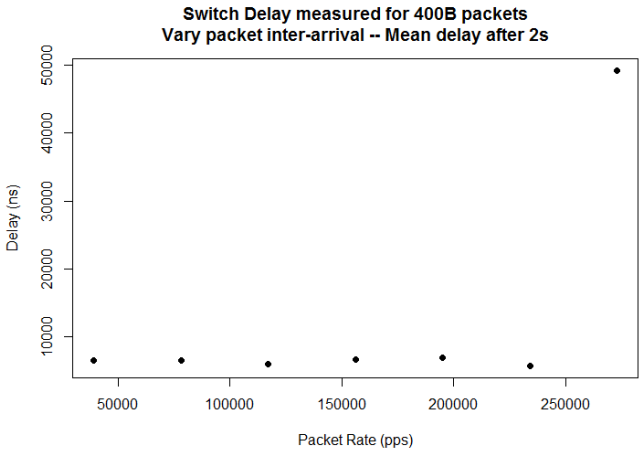
Beyond the first section, the average difference between the direct and via-switch measurements can be compared to the switch service time measured in Chapter 4. Below 30000pps the median delay measured is 4.68 microseconds. The delay measured in Chapter 4 for 400 byte packets is 6.92 microseconds. This is not satisfactory. This can be taken as an indication this method of measurement is not suitable for the microsecond range required here. Controller path delay, which is on the order of microseconds, is more suitable to be measured in this way.

#### Results of Extra Measurements

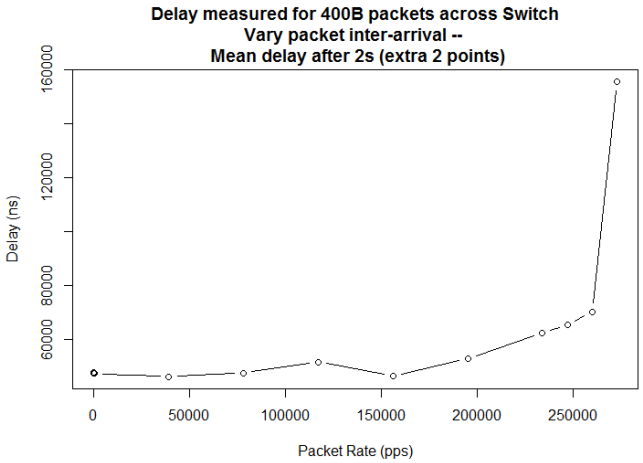
Extra measurements at 750 and 875 Mbps, visible in only Figure 5.3 (c), show the final section is a continuous curve, continuing the exponential trend. It is not a fourth section starting a new curve. Note: These extra two measurements were performed as a single iteration, and are only used to check the trend of the final section.




(a) Direct and Data Path




(b) Difference between Direct and Data Path




(c) Difference – Extra Data Points

Figure 5.3: Average delay of various arrival rates.

# Chapter 6

# Experimental Results

This chapter describes the experimental performance results from a real SDN network as network parameters are changed for the purposes of validating the results of existing SDN queueing models. In Chapter 7 these results are compared to the results of the existing queueing models.

The variables measured are the sojourn time and packet loss. These measurements will be taken across a parameter space of packet sizes, arrival rates and $P_{nf}$ ratios. Accompanying these results will be a description of the measurement procedure and some insight into what the results mean for the test network.

The measurements are performed across two ranges of arrival rate parameters, each paired with the delay of one of the two paths packets can take in a reactive network.

## 6.1 Two Parameter Ranges

Based on the exploration in Chapter 5, two parameter ranges are decided upon. Two ranges were used as a result of the control plane delays being several magnitudes higher than the data plane delays.

When the rate of packets on the control plane is too high significant amounts of traffic will be lost. This is a reactive scenario, so the rate of

packets sent to the control plane is a function of the arrival of new flows. As in previous work [12, 14], the probability of new flows is represented as $P_{nf}$.

### 6.1.1 Range One

Range one covers a higher packet rate at the expense of a lower $P_{nf}$ value, to avoid excessive packet loss on the controller path. This range allows more of the switches available forwarding capacity to be utilised.

An end-to-end delay method is used to measure this range, so the same experiment must be run twice. First to measure the delay between the two hosts (direct), and then to measure the sojourn over the network equipment (via-switch). The difference between these two will leave the average delay of the SDN network. This is required as the parameter investigation in Chapter 5 shows the direct host-to-host delay has some state-dependent delay, thus a single value for delay cannot be applied universally to all inter-arrival rates or packet sizes. The high rates do not allow the switch to be used in the network event method.

Table 6.1 shows the parameters in use for this range.

| | | | | | | |
|---|---|---|---|---|---|---|
| **Packet Sizes (Bytes)** | 100 | 200 | 600 | 1000 | 1400 | |
| **$P_{nf}$ Values** | 0.005 | 0.01 | 0.02 | | | |
| **Packet Rates (pps)** | 5000 | 10000 | 15000 | 20000 | 25000 | 30000 |
| **Sample Duration (s)** | 260 | 135 | 95 | 75 | 62 | 54 |

Table 6.1: Range One Parameters

Each sample contains 1.26 million observations, with the first 2 seconds removed to allow the system to warm up. A higher inter-arrival rate allows more observations to be collected in a small amount of time.

One drawback of this range is that while 30000 pps is a low bitrate for 100B packets (24 Mbps), the bitrate of 1400B packets is one third of

the maximum capacity of the link (336 Mbps), so 1400 byte packets will stress the system more than other packets and approach the area where the maximum bitrate of the measurement hosts and link are reached. The threshold of data rates shown to be free from the anomalous delay packets (Section 5.2.2) is 1Mbps, above which the difference between via-switch and direct delays can be calculated.

### 6.1.2 Range Two

For this range the arrival rate of traffic is more sympathetic to a slow controller path, allowing a wider range of $P_{nf}$ values to be explored.

Based on the parameter exploration, the inter-arrival times of range two lay partially within the area where the anomalous delay packets could be expected to occur. Because of this, the end-to-end method of delay measurement cannot be used. However, the packet rate is low enough that the network event method can be used.

Similar to the method of measuring service times across the controller path, the packets entering the switch are cloned and sent to the sink. One packet re-enters the switch and the other is passed to the sink. The difference between the cloned packet and network-delayed packet is then used to calculate sojourn time.

Table 6.2 shows the parameters in use for this range.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Packet Sizes (Bytes)** | 100 | 200 | 600 | 1000 | 1400 | | | | | |
| **$P_{nf}$ Values** | 0.01 | 0.04 | 0.25 | 0.50 | | | | | | |
| **Packet Rates (pps)** | 20 | 40 | 60 | 80 | 100 | 120 | 140 | 160 | 180 | 200 |
| **Sample Duration (s)** | 270 | 150 | 100 | 80 | 70 | 55 | 47 | 41 | 37 | 33 |

Table 6.2: Range Two Parameters

Each sample contains at least 4000 observations, and is repeated seven times. The average of these repetitions for each set of parameters is then calculated.

## 6.2 Method

The same experiment is performed for both parameter ranges. The differences between them consist of only the method used to record delays and the parameter range used, as per Tables 6.1 and 6.2.

A PCAP file is created which contains traffic exhibiting a particular $P_{nf}$, expressed in transport layer port numbers. The seed used for this random process is recorded and changed each iteration and trial. Packets from the PCAP file are sent from the source host with exponentially distributed inter-packet times, following the mean packet rate parameter, using the tool evaluated in Section 3.4.2.

As described in Section 3.5 the switch has two rules to split traffic into data and controller paths, based on transport layer port number. One rule matching packets destined for the controller, emulating the arrival of a new flow. A second rule matching packets to be forwarded on to the sink.

In this test environment a new flow is signalled by a transport layer port number. The PCAP file and the switch flow rules are configured as such.

Again, as with the controller path service time experiment (see Section 4.2.1) a $flow\_mod$ message is sent for each packet visiting the controller, and the flow added by this uses priority to prevent it from being matched and replaces existing copies of this rule.

### Realism

When TCP is used in a real network, if the *SYN* packet is lost, the remaining packets are not sent. In this set of experiments the flows used are stateless and the packets will continue to be sent ignorant of the fate of the "first" packet.

## 6.3 Range One Results

Because of the low $P_{nf}$ parameters, the filter interval used to remove outliers from the data must not be too narrow to remove this feature in the measurements. Thus a filter range of 99.99% (compared to minimum $P_{nf}$ of 0.5%) was selected to remove extreme outliers.

### 6.3.1 Sojourn Distribution

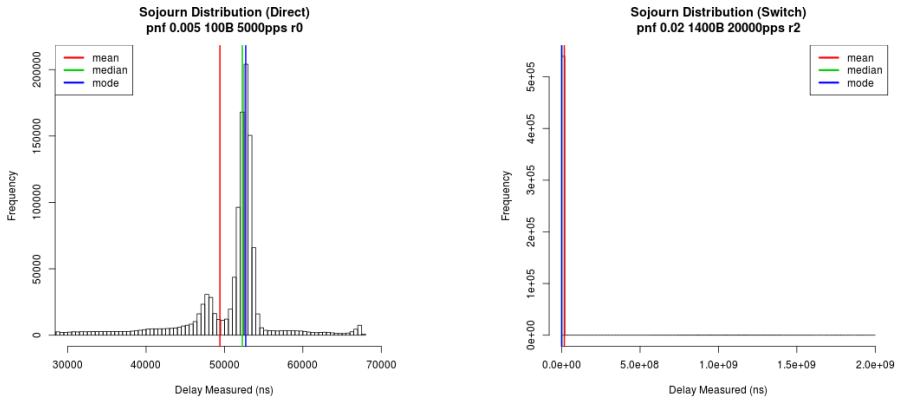
Presented in Figure 6.1 is a sample of the sojourn time distributions of the direct and via-switch delay results, of which the difference is used to calculate the average sojourn delay for range one.

The distribution of the direct results show a bimodal trend. The distribution of the results via the switch is unclear when considering the complete set of data recorded. So, as in Figure 6.2, the data is considered in lower, middle and upper sections.

The lower section shows the shape of the mode visible at the far left of Figure 6.1. The frequency of this section overshadows the rest of the distribution, and sits at around 50 microseconds. The middle section clusters around 5 milliseconds, spreading up to 20 milliseconds. The upper section ranges from half a second to two seconds.

Relating the three sections presented in Figure 6.2 to the behaviour of the network, the lower section represents the packets that follow the data path. On account of the magnitude of the middle section, and the magnitude of the clustered regular intervals present on the controller path, the middle section possibly contains the packets that were caught in the intervals.

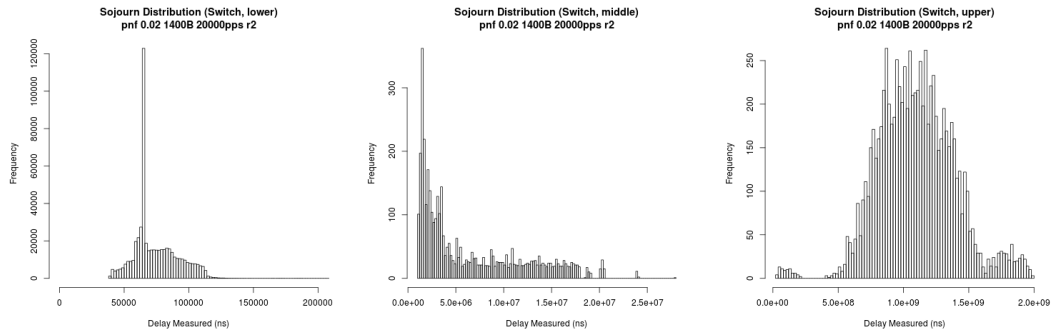
The upper section present results on the order of seconds. These delays are very significant, being around a second. Interestingly this corresponds to the duration Bifulco *et al.* [5] reports from TCAM to be updated, so could indicate this is caused by updating the flow table. The *via switch* delay distributions shown here include the overhead of the hosts' sending and




(a) Direct (b) Via switch

Figure 6.1: Distribution of direct and via switch sojourn times

receiving, which is removed in the next section for the averages presented.




(a) Lower (b) Middle (c) Upper

Figure 6.2: Distribution of sojourn times via switch

### 6.3.2 Average Sojourn Time

Figure 6.3 presents the mean sojourn and mean packet loss of the range one experiments, separated by packet size and $P_{nf}$.

Immediately apparent is the decreasing sojourn time as the packet rate

is increased. This corresponds to a greater rate of packet loss as the rate increases. However, through the range of packet rates used this does not match the increase in packets entering the network.

Notice that while the trend of the measured sojourn time is decreasing with increasing packet rate the magnitude of the delay is increasing as the $P_{nf}$ increases. As packet size increases, the sojourn time decreases and the packet loss increases.

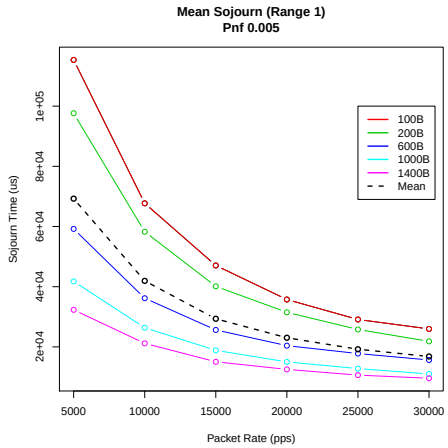
### 6.3.3 Discussion

The downward trend, as stated earlier, is unexpected. However, it can be explained by considering which packets are being lost.

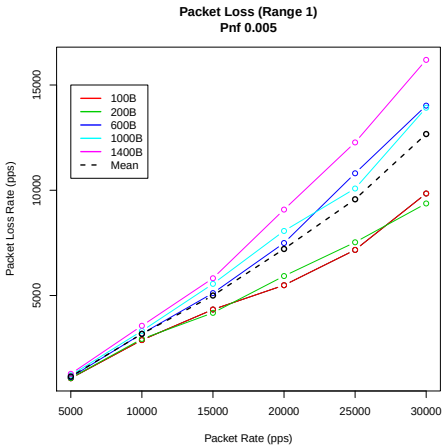
The packets passing through only the switch aren't significantly affected by the delays across the controller path – although there is certainly some interference as discussed in Section 5.1. The packets being lost are those in the middle and upper sections of the delay distribution. Compared to the lower section, these packets are few in number, and experience much greater delay. When the extreme values are lost the average delay across the entire distribution decreases significantly.

The increase in packet loss is interesting. It is known that the interference causes loss to occur on the data plane when it should not. An increase in the arrival rate of packets of 5000pps results in losses of almost half that when only 5% of packets are destined for the controller. This raises a concern as to the impact of this interference with high rates of traffic.

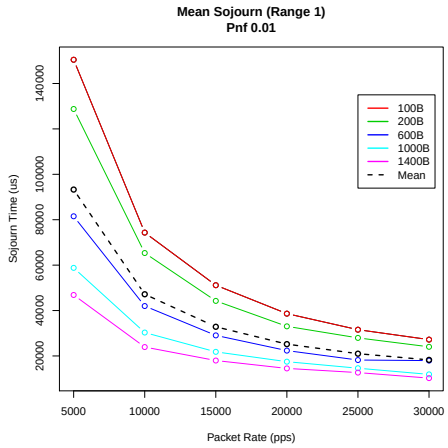
The trends in packet size can be explained when considering larger packets will occupy more space in a switch's buffer, filling it faster than smaller packets sent at the same rate. Effectively, the queue will be shorter for larger packets, and packet loss will begin to occur at a lower packet rate.



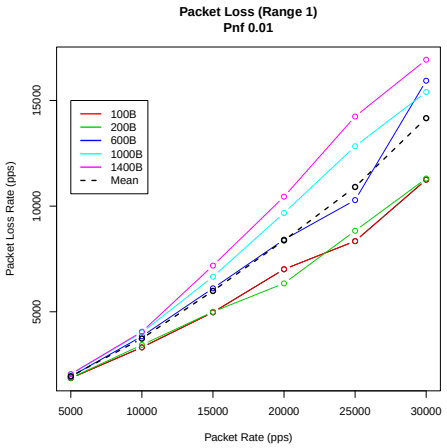

(a) $P_{nf}$ = 0.005



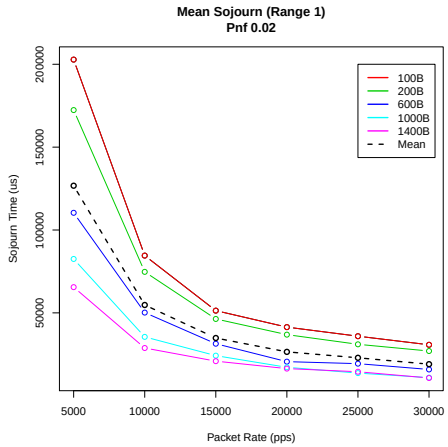

(b) $P_{nf}$ = 0.005



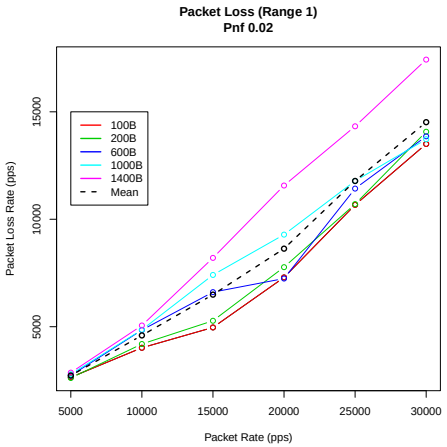

(c) $P_{nf}$ = 0.01




(d) $P_{nf}$ = 0.01




(e) $P_{nf}$ = 0.02




(f) $P_{nf}$ = 0.02

Figure 6.3: Average sojourn and packet loss of range one

## 6.4 Range Two Results

### 6.4.1 Sojourn Distribution

A sample of the sojourn distribution for parameter range two is shown in Figure 6.4. The lower and upper sections are shown separately in Figure 6.5. When these sections are combined, the separate service processes of the paths the packets take are not obvious.

Because of the smaller range displayed, the lower section shows clear quantisation of measured times. As hardware timestamps are used, almost no variation is shown for the average switch sojourn time visible in the two discrete bars of Figure 6.5 (a). As packet size and rate increases this group increases, as seen in the median of Figure 6.8.

The upper section of the sojourn time distribution shown in Figure 6.5 (b) is much smaller than the lower section, containing only results which are caused by $P_{nf}$ proportion of the data. Visible here are the intervals of four milliseconds, discussed in Chapter 5.

From the sample of data, a larger $P_{nf}$ results in a much greater spread in the upper section.

### 6.4.2 Average Sojourn Time

The results of mean sojourn time are shown in Figure 6.7. The median sojourn time is shown in Figure 6.8. The mean packet loss for each $P_{nf}$ is shown in Figure 6.6.

Medians are less affected by outliers in data. For the lower $P_{nf}$ values the percent of values passing through the controller are very low. It can be seen that until 25% or more of the packets are travelling through the controller path the median sojourn is not affected. Only the delay of packets passing through the data path influence the median sojourn time.

Unlike range one, parameter range two's results show the expected increasing sojourn as packet rate increases. Evident in Figures 6.7 (c) and

(d), the average sojourn of the $P_{nf}$ 0.25 and 0.5 experiments increases exponentially. $P_{nf}$ 0.5 shows the most interesting trend of a flattening out of the sojourn time after the section of exponential increase.

Corresponding to the sections of exponential increase in sojourn time is a dramatic increase in the average results of packet loss. The results show an increase close to the increase in arrival rate of traffic. The packet loss here is expressed as an average rate over the duration of the experiments instead of a cumulative count. This makes comparison between each experiment and to the queueing models in Chapter 7 easier.

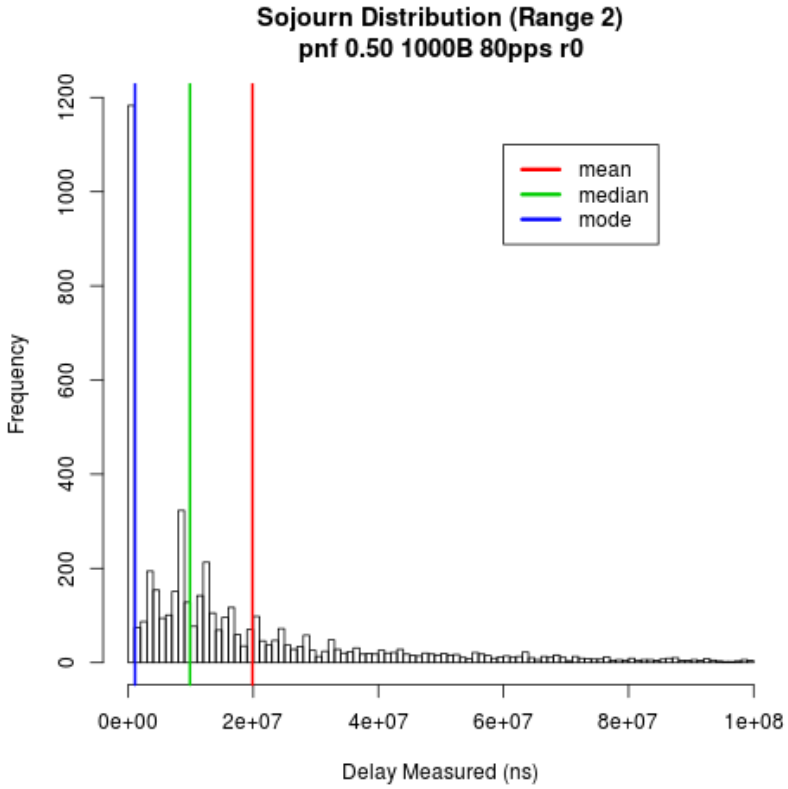



Figure 6.4: Distribution of sojourn delay for range two

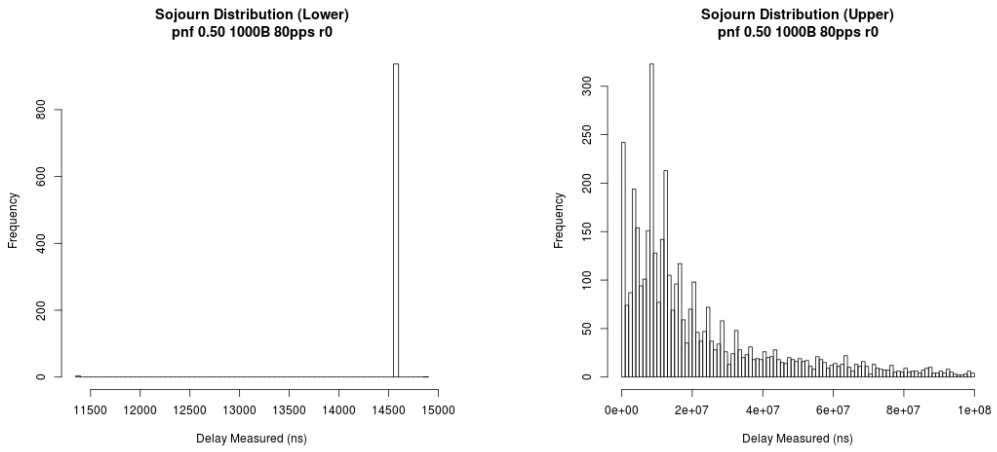
### 6.4.3 Discussion

An exponential increase in sojourn time indicates that the ratio of arrival and service rates are close to being equal. The rate of traffic to the controller path is controlled by $P_{nf}$. In Figure 6.7 it is clear that the point of exponential increase is changing as a function of the $P_{nf}$. Thus it is the controller path's service time which is being met by the increasing arrival rate of packets. The combination of the increase in packet loss and increasing sojourn time which then flattens out indicates this is caused by a buffer filling up on the controller path. This is seen in Figure 5.1

The effect of the controller path places an estimated service rate of the controller to be between 100-200 pps. This is a service time of 5-10 milliseconds, which is close to the time measured in Chapter 4. Being close to the time in Chapter 4 is an indication that the queueing models may be able to provide reasonable predicted results. Some state-dependent service times on the controller path, where the processing time is shorter for faster arrival rates, could account for any differences between the estimate here and the measured times in Chapter 4.

The medians in Figure 6.8 (a) and (b) show that the additional queueing delay, seen in the increase of the mean, is restricted to the controller path. On average, there is no queueing occurring for regular traffic on the data path. It's not until more than 4% of packets are following the controller path that the median is affected enough. The controller path interfering with the data path is the cause for this, and not regular switch path queueing.

As expected with the lower packet rates in this range, only the high $P_{nf}$ results show significant signs of packet loss.




(a) sojourn range2 dist lower

(b) sojourn range2 dist upper

Figure 6.5: Distribution of sojourn times in range two

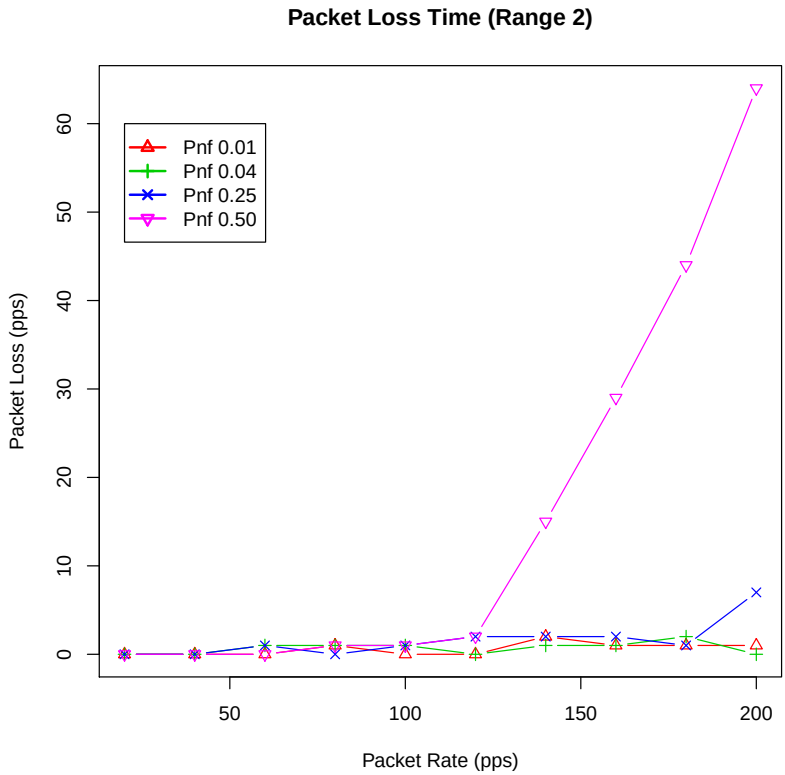
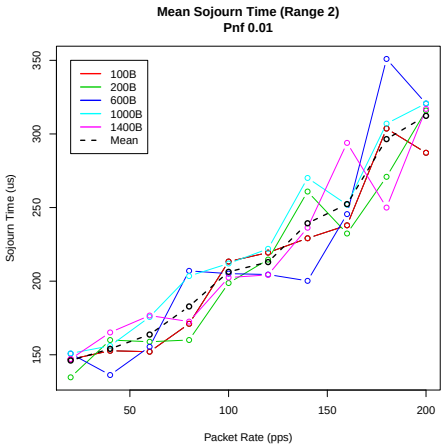
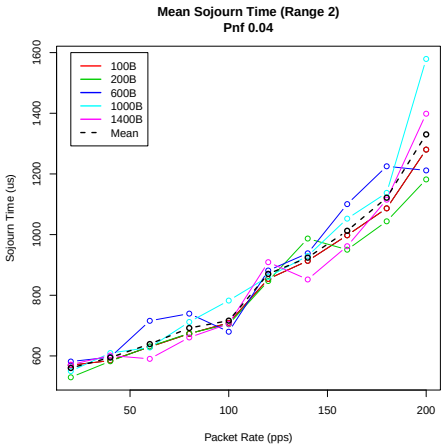


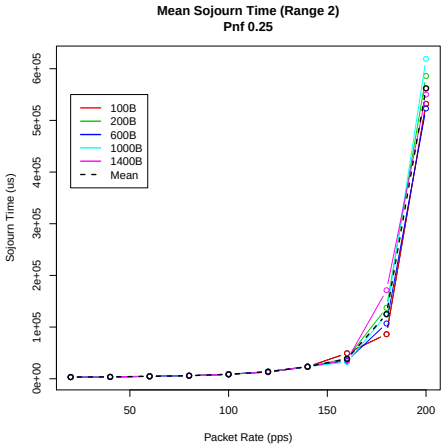

Figure 6.6: Average Packet Loss of each $P_{nf}$



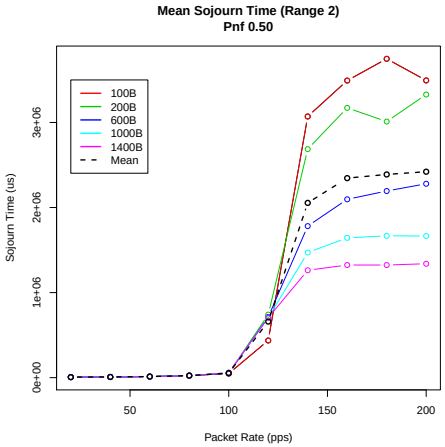

(a) $P_{nf}$ 0.01




(b) $P_{nf}$ 0.04

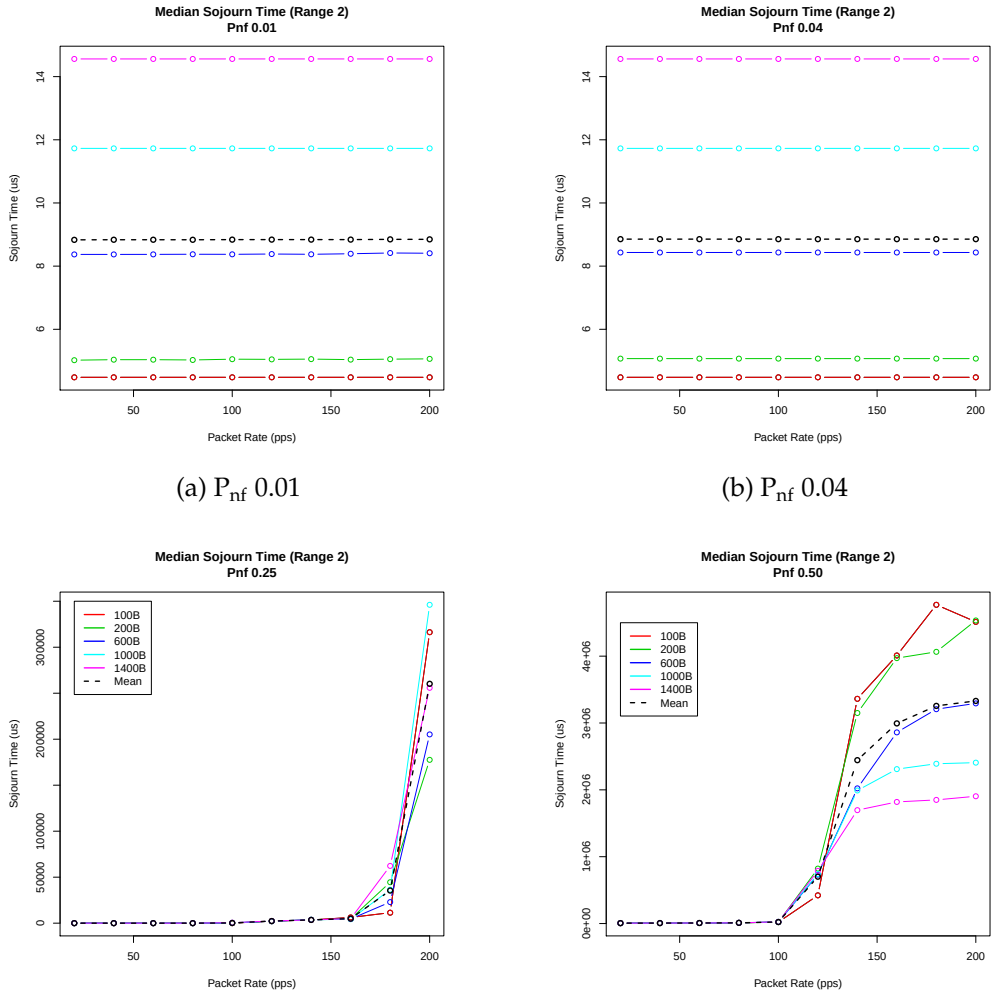



(c) $P_{nf}$ 0.25




(d) $P_{nf}$ 0.50

Figure 6.7: Mean Sojourn time (Range 2), the effect of rate and packet size on sojourn time




(a) $P_{nf}$ 0.01

(b) $P_{nf}$ 0.04

(c) $P_{nf}$ 0.25

(d) $P_{nf}$ 0.50

Figure 6.8: Median Sojourn time (Range 2), the effect of rate and packet size on sojourn time

### 6.4.4 Effective $P_{nf}$ Value

Because of the additional packets which suffer the delay of the control plane, it begs the question of how this affects the overall average delay. One way to quantify this is to compare the proportion of packets that were sent to the controller with those that suffered a delay greater than the expected threshold for dataplane traffic.

This metric will be called the *effective* $P_{nf}$ and is calculated as per equation 6.1, given a threshold $T$ is used to classify packet delay.

$$\frac{n_{effCtrl}}{n_{effData} + n_{effCtrl}} \tag{6.1}$$

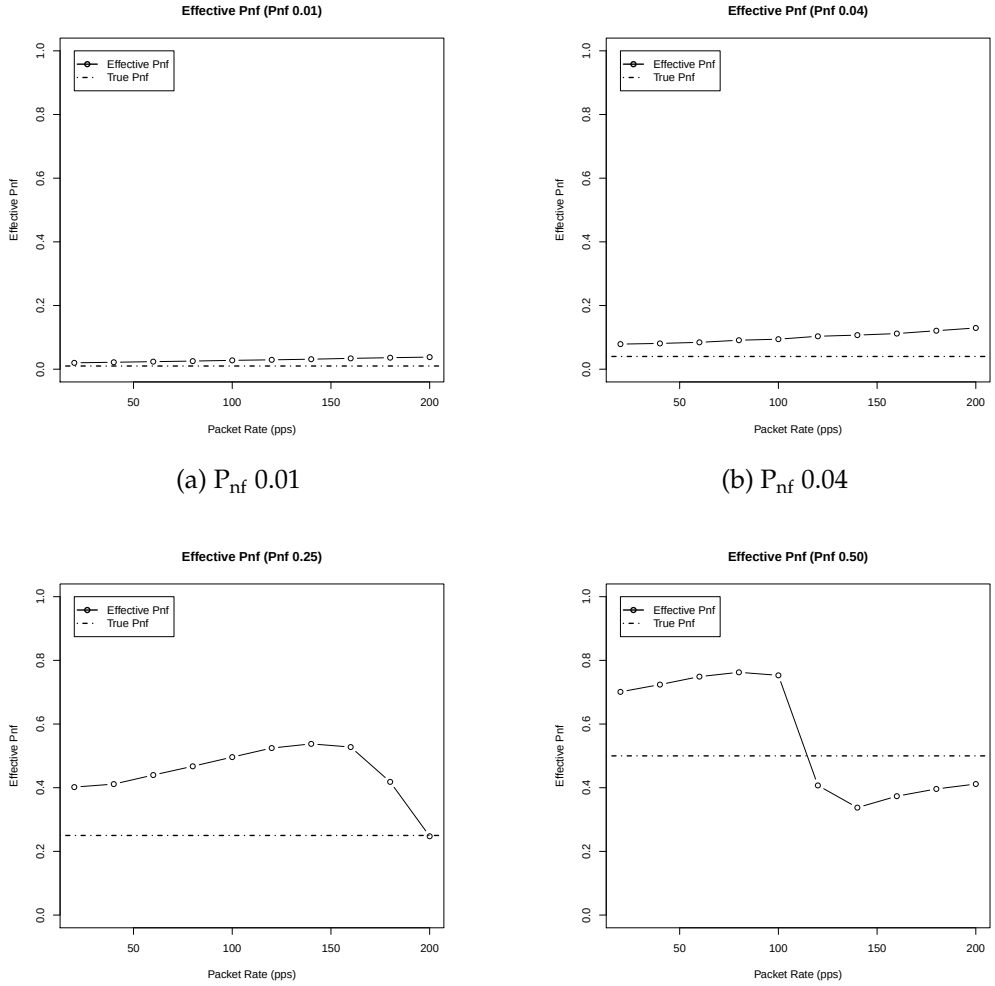
Greater than $T$ will be *effective controller path* traffic ($n_{effCtrl}$). Less than $T$ will be *effective data path* traffic ($n_{effData}$).

Range 1 experiments suffer significant packet loss, making it impractical to consider which packets were supposed to be lost and those that were not, so this question cannot be asked of that data. Range 2 suffers comparatively little loss, so is used here.

These effective $P_{nf}$ results are presented in Figure 6.9 as the mean across the packet sizes of each packet rate.

In general the trend is that the effective $P_{nf}$ is greater than the true $P_{nf}$. This is not the case for $P_{nf}$ 0.25 and 0.5 as the packet rate passes 100 pps, where it becomes less than the true $P_{nf}$. Comparing these results to the average packet loss rate in Figure 6.6, it appears packet loss is the cause of this.

Because of packet loss, the overall effect on delay is mitigated. Up until the point of packet loss, the impact of additional packets being delayed is quite significant – on average doubling the delay expected of $P_{nf}$ 0.01 and 0.04 and increasing that of 0.25 by 77%.




(a) $P_{nf}$ 0.01

(b) $P_{nf}$ 0.04

(c) $P_{nf}$ 0.25

(d) $P_{nf}$ 0.50

Figure 6.9: Effective $P_{nf}$ results

# Chapter 7

# Compare Measured and Modelled Results

The analytical results of the four models are presented here using the service rate parameters obtained in chapter 4. The experimental results of a real SDN network are used to determine the accuracy of the modelled results. The results are compared and differences quantified and described.

The following models are compared to the experimental results:

- M/M/1 model, representing the base-case
- M/M/1/K model, representing the base-case with a finite queue
- Jarschel's model
- Mahmood's single switch model

This chapter introduces two additional queueing models, these are two of the most fundamental queueing models. M/M/1/$\infty$ and M/M/1/K. In this Chapter they will be referred to as the *base case models*. These represent a single queue with Poisson arrivals and exponentially distributed service times. The difference between them is the queue capacity, where the former has no limit and the latter has $K - 1$ spaces in its queue. They represent the simplest cases of a queueing model. Section 7.1 details applying the measured service times to the base case models.

These models can be placed into two categories. The first category is models with an infinite capacity, these are the M/M/1 Model and Mahmood's model. The second category is models with finite capacity, the M/M/1/K Model and Jarschel's model.

Based on the s-shaped trend seen in the results of Chapter 6 it is expected that the finite capacity category of queueing models will offer a better comparison to the experimental results. Infinite capacity models will not be able to produce results for packet loss.

For the discussion of these results the load of the queueing model is discussed. The load is $\rho$, where $\rho = \frac{\lambda}{\mu}$; the rate of traffic arrivals ($\lambda$) divided by the service rate ($\mu$).

Note that the region where Mahmood's model produces invalid results is removed from the analysis ($\rho > 1$).

### Scaled Difference

The difference between the measured and calculated results can be tricky to compare directly. The *scaled difference* used here is calculated by equation 7.1. By dividing the difference by the measured result the "error" between the modelled results and measured results can be compared. A scaled difference of zero would indicate the model matches the experimental results at that point.

$$\frac{(X_{measured} - X_{modelled})}{X_{measured}} \tag{7.1}$$

### Model Parameters

Only the results from the range two measurements are compared to the experimental results here (see Table 6.2). The results of range one are taken at a point where the models are not stable, and $\rho$ is much greater than one. Thus, most models produce invalid results in the parameters of range one. Range two offers a suitable set of data for the comparisons in this chapter.

Service time parameters for the models are taken as an average of the result in tables 4.1 and 4.3. The service rate, $\mu$ parameter, is equal to $\frac{1}{service\_time}$.

For finite capacity models the queue capacity parameter is estimated from the packet loss results of Chapter 6 to be roughly 200 packets.

## 7.1 Using the Base Models

The base models used here have the facility for representing only a single queue and server. To apply the two service processes occurring in a SDN network to the single-queue base models the service time of each path is combined using a $P_{nf}$-weighted average. This calculation is shown in equation 7.2, where the $\mu_{controller\_path}$ is the inverse of the sum of the three average controller path service times.

$$\mu_{base\_model} = (1 - P_{pnf}) \cdot \mu_{switch\_path} + P_{nf} \cdot \mu_{controller\_path} \tag{7.2}$$

This simple process allows the results of the SDN models and the results of the base queueing models to be compared to the experimental results.

## 7.2 Sojourn Time

The graphs of $P_{nf}$ 0.01 and 0.04 are displayed separately from those of $P_{nf}$ 0.25 and 0.50 for clarity due to the differences in magnitude.

The sojourn times of the infinite capacity queueing models are shown in Figure 7.1 and the sojourn times of the finite capacity queueing models are shown in Figure 7.2. To compare these two models to the experimental results the scaled sojourn differences are shown in Figure 7.3.

The $P_{nf}$ 0.01 and 0.04 results for all models are nearly identical. The interesting points are those in $P_{nf}$ 0.25 and 0.50.

### 7.2.1 Arrival and Processing Rates are Equal

As expected of infinite capacity models, the sojourn time increases exponentially leading up to the point where the arrival rate on the controller path is equal to the service rate of the controller path ($\rho = 1$). Similarly, at the same point the sojourn time of the finite capacity models flatten off.

Comparing the behaviour at the point of the models where $\rho = 1$ to the measured results show that the modelled results are prematurely increasing exponentially. Seen in Figure 7.3 this causes the biggest deviation of the modelled results for both infinite and finite capacity models.

### 7.2.2 General Trend

Looking at the $P_{nf}$ 0.01 and 0.04, their similarity between both infinite and finite capacity models hints at a key difference between the modelled and experimental results. The modelled results start at a higher sojourn, and increase at a lower rate than the measured results. The measured sojourn results intersect with the modelled results until the point where $\rho = 1$.

This difference in trend is also seen in the steady increase of the scaled difference in Figure 7.3. The scaling of this comparison method should eliminate the increasing trend of the data, leaving only the trend in differences between them. Note too that this increase in scaled difference is common to all the $P_{nf}$ values. Interestingly, even when the $P_{nf}$ 0.50 difference increases due to the premature exponential increase it then returns to the same line followed by other $P_{nf}$ values. Given this fact, the effective $P_{nf}$ described in Section 6.4.4 cannot be the cause of this trend either.

### 7.2.3 Scaled Difference

Comparing the mean and sum of the scaled differences, a numerical measure of how well the model represents the measured results can be calculated. The sum and mean of the absolute values of the scaled difference

Table 7.1: Scaled difference comparison

| | **M/M/1** | **M/M/1/K** | **Jarschel** | **Mahmood** |
|---|---|---|---|---|
| **Sum** | 778998 | 775462 | 14.60791 | 14.73515 |
| **Mean** | 17311 | 17232 | 0.417369 | 0.4210042 |

for the sojourn time of four models is presented in Table 7.1.

To compare Jarschel and M/M/1/K fairly, the results that are invalid in the infinite category models are also removed from the finite models. As the scaled difference represents the distance from the modelled results, a smaller number is better.

Jarschel's is the best using this method having the lowest average and total scaled difference. Visually however all the models are very similar. Without better mathematical tools and more data it would be difficult to decide which is the best.

## 7.3 Packet Loss

The packet loss rate of the finite capacity queueing models are shown in Figure 7.4. The scaled difference between this and the experimental results is shown in Figure 7.5.

As with the sojourn results, the packet loss starts to increase ahead of the packet loss of the experimental results. This occurs at the same point as the sojourn times increase too.

There isn't much information in the plots for $P_{nf}$ 0.01 and 0.04. The model predicts a rate of loss which is non-zero but less than one. The experimental results shows a very low rate of packet loss, but not zero.

The gradient on the packet loss line of the $P_{nf}$ 0.50 modelled results appears stationary. However the gradient of packet loss line seen in the experimental results is increasing. The gradient of the M/M/1/K results changes in the wrong direction. The scaled differences show this too,

M/M/1/K being slightly worse, though similar in trend to Jarschel's model's results.

## 7.4 Discussion of Comparison

Both categories show very similar results at the first intersection. For $P_{nf}$ values 0.01, 0.04 and 0.25 there is an intersection between the experimental and modelled results in the lower packet rate range. For a $P_{nf}$ of 0.50 this intersection occurs much later. Here it is seen that the experimental sojourn results begin lower than the modelled results, but have a more positive gradient. Thus they cross in the lower sections of the sojourn curve.

For the infinite capacity models, this is the only intersection, and the modelled sojourn results remain higher from that point until $\rho$ is equal to one. Infinite capacity models are only valid for values of $\rho$ less than one.

### 7.4.1 Accounting For Differences

This section outlines the possible causes of the differences between measured and analytical results.

The effect of interference between the controller and data paths of packets must be mentioned. It is responsible for the upper-most sections experimental results' distributions. The possible cause of this is discussed in Section 5.1. This influences the distribution of service time of the controller path, and thus could have the effect seen here on the sojourn time.

Both Jarschel's and Mahmood's models show similar overall trends different to the trend seen in the measured results. Using Figure 2.3 as a guide, the elements these models have in common are:

- (A) Inter-arrival times of packets are assumed to be exponential.
- (B) It is assumed that only TCP traffic is present in the network.

- (E) Purely reactive flow rules.
- (F) The switch is modelled as M/M/1/$\infty$, modelling the time between ingress traffic events and service times as the time between events in a Poisson process.
- (G) Queue is shared between ingress data traffic and traffic from the controller.
- (N) Controller egress traffic is sent to the one switch queue.
- (C) New flows are represented by $P_{nf}$ at the switch deciding to sent packets to the controller.

Of these features, A, B, C, and E are all met by the experiment. F, G and N are the only features not explicitly met by the experiment. G and N are the same here. These features are considered below.

- (F) The switch in the experiment is finite capacity, but the capacity is never pushed. The service times don't appear to follow an exponential distribution and this affects all the results. Ingress traffic is exponential.
- (G) The queue for the controller is on a separate port to the ingress data traffic.

In this experiment the port the controller connects to the switch with is dedicated to the controller. However the service times of the data path may not be exponential.

There may be a third queueing process separate from the two queues of the SDN models (See Section 5.2.2). It is the queue in the switch responsible for handling communication with the controller. This is the point in the network model that also concerns all the possible causes for differences listed here. This could indicate that this is an area that should be modelled differently to bring the modelled results closer to experimental.

The premature rise in sojourn time is related to the overall trend of the model. However it could be that the parameter used to represent the service time of the controller is not correct. Because of the clustering in the controller path service time distributions, the controller's service time certainly doesn't follow an exponential distribution. This could account for the *late* increase in sojourn seen in the experimental results.

Specifically for the Jarschel's sojourn result, the queue capacity parameter could influence how close the model can get. However without changing the service time or $P_{nf}$ the overall trend cannot match up. This applies to the packet loss too. The trend of the packet loss line is linear for Jarschel's packet loss results, and increasing for the experimental results. This indicates there is a process not incorporated within the model that is causing packet loss. Again, this difference could be the result of the third queueing process presently missing from the models.

### 7.4.2 Base and SDN Model Similarity

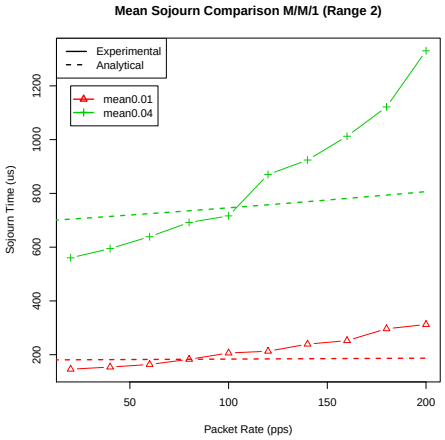
While not a comparison to the experimental results, when comparing the two analytical models in each category to one another, they appear to be very similar. Comparing them numerically, dividing each result of the SDN model result by the corresponding base model result, it is found that they are indeed almost exactly the same around 50 pps. Further above or below this within the arrival rates shown and they deviate on the magnitude of $10^{-3}$. For this range of parameters the models in each category are equivalent.

Because of this, when calculating the differences for each category from the experimental results the base version is redundant and omitted.

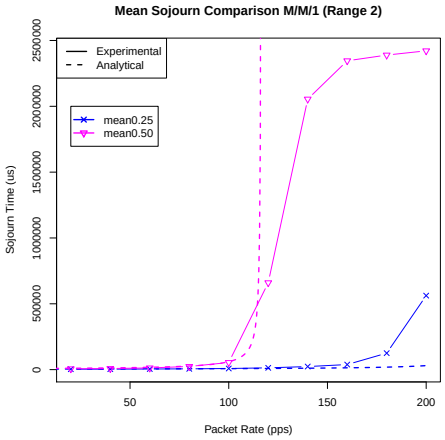
For packet loss, the base and SDN models have different results.

## 7.5 Suggestions for improving models

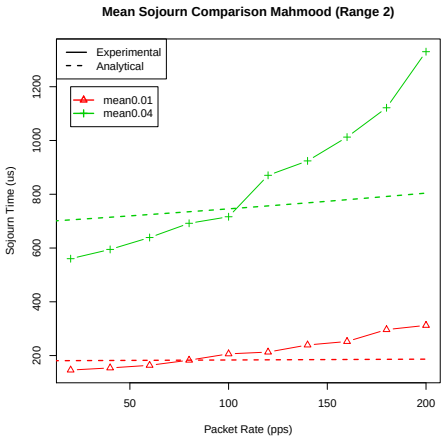
The effect of test equipment on the measured results makes conclusions difficult. Despite this the indication from the discussion in the previous section indicate there could be some value in including a third queueing process for the controller interface component of the switch in the model.



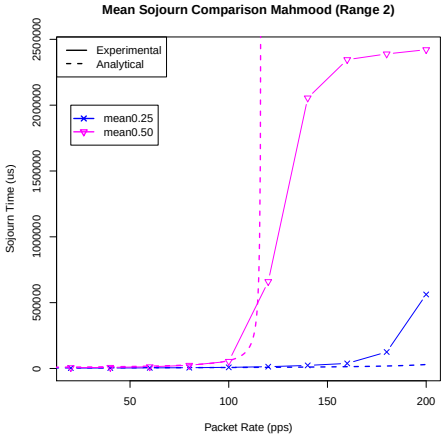

(a) M/M/1 Model $P_{nf}$ 0.01 and 0.04




(b) M/M/1 Model $P_{nf}$ 0.25 and 0.50

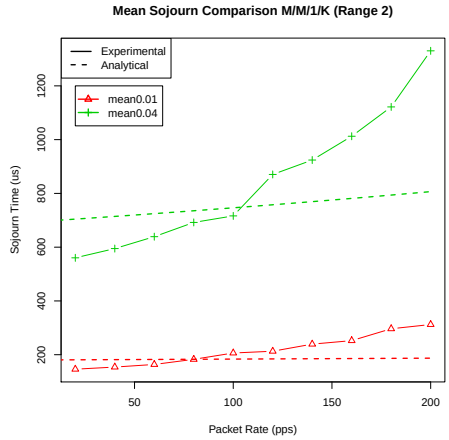



(c) Mahmood's Model $P_{nf}$ 0.01 and 0.04

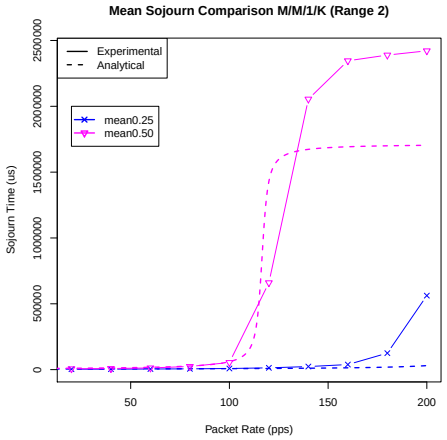



(d) Mahmood's Model $P_{nf}$ 0.25 and 0.50

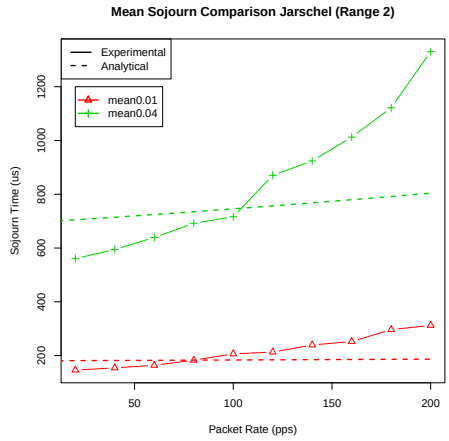
Figure 7.1: Comparison of sojourn time for infinite capacity models



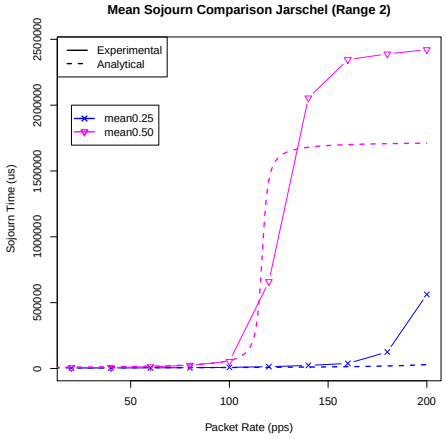

(a) M/M/1/K Model




(b) M/M/1/K Model

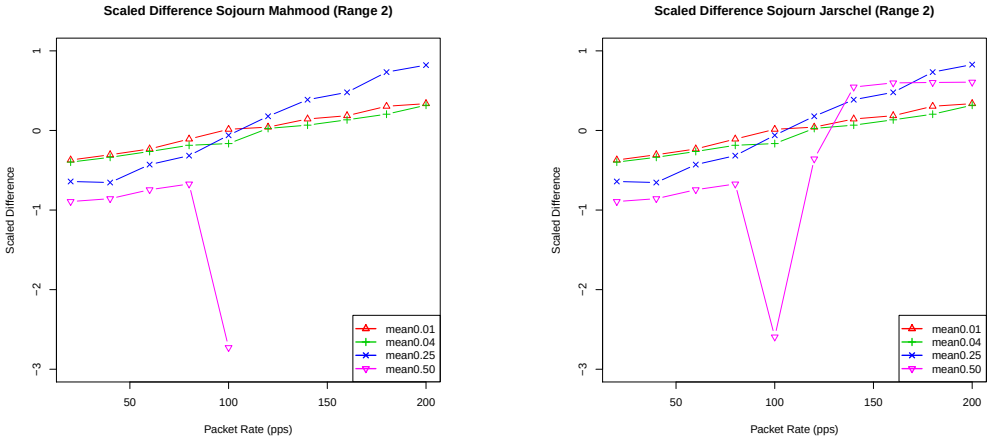



(c) Jarschel's Model




(d) Jarschel's Model

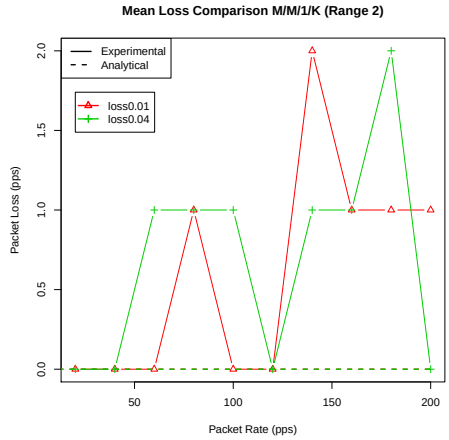
Figure 7.2: Comparison of sojourn time for finite capacity models



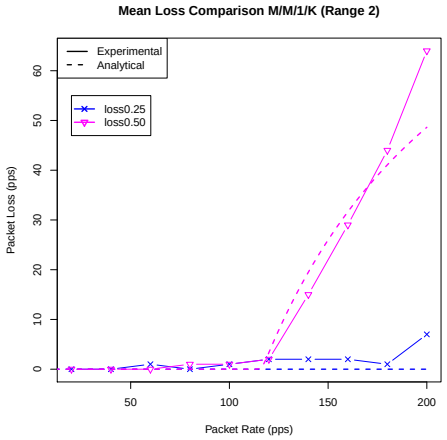

(a) Mahmood's Model

(b) Jarschel's Model

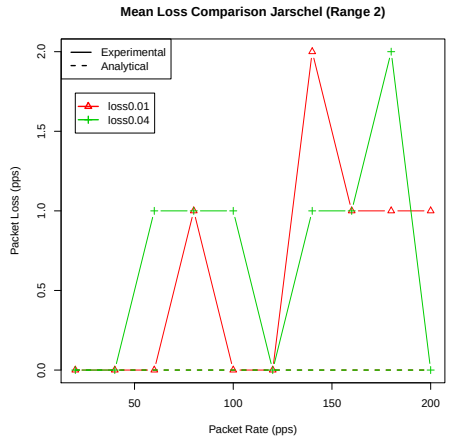
Figure 7.3: Scaled difference between SDN analytical models and experimental sojourn results



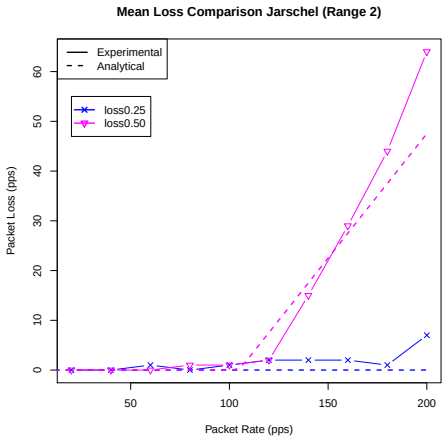

(a) M/M/1/K Model $P_{nf}$ 0.01 and 0.04




(b) M/M/1/K Model $P_{nf}$ 0.25 and 0.50

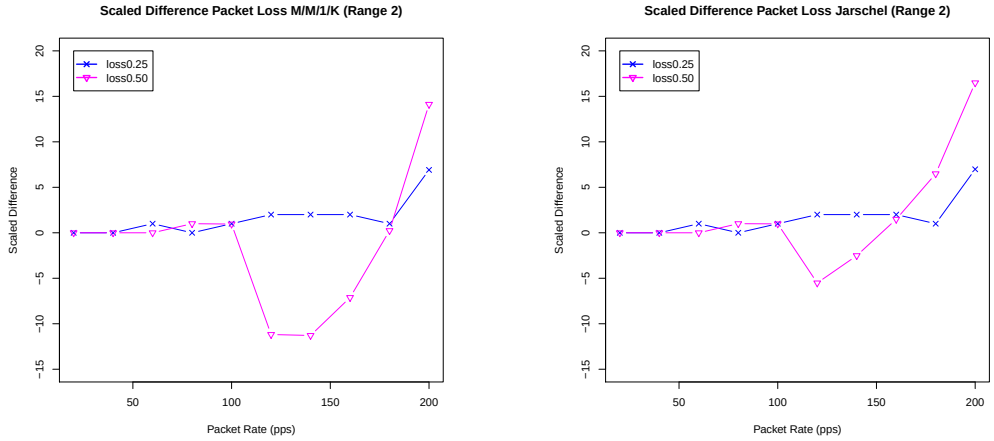



(c) Jarschel's Model $P_{nf}$ 0.01 and 0.04




(d) Jarschel's Model $P_{nf}$ 0.25 and 0.50

Figure 7.4: Comparison of packet loss rate for finite capacity models




(a) M/M/1/k Model

(b) Jarschel's Model

Figure 7.5: Scaled difference between SDN analytical models and experimental packet loss results

# Chapter 8

# Conclusions

This chapter summarises the work presented in this thesis by highlighting the notable findings. Features that future work might consider are also described here.

## 8.1 Thesis Purposes

The first purposes of this thesis are to quantify and compare the accuracy of the existing queueing models with experimental results. The experimental results from Chapter 6 are compared to the existing queueing models from Jarschel *et at.* and Mahmood *et at.* in Chapter 7. Through the use of the method of scaled difference, it was found that Jarschel *et at.*'s model is slightly more accurate than the rest at modelling the traffic of the network measured.

The second purpose of this thesis was to highlight the features of an SDN network which could improve the results of the analytical models based on literature and experimental observation. Based on sojourn and packet loss results and examination of the features of the existing SDN queueing models, the discussion in Chapter 7 directs potential improvements to the existing queueing models to be around a third queueing process within the switch. This component communicates with the controller

and separates the forwarding and controller communication queues. The effects of this missing component need to be understood to improve the accuracy of SDN queueing models.

#### Behavioural Quirks of Hardware

Because of the open nature of SDN and OpenFlow, manufacturers of switch equipment need to integrate it with their products themselves. This can lead to shortcomings in the implementation. As seen with both the ZodiacFx (Chapter 2) and the AT-510x switches (Chapter 5), the realised behaviour and performance is not what was expected.

In terms of network scenario, based on the results a purely reactive scenario with fine-grained flows may not be best for SDN networks. Network administrators must be mindful of the limits of their network hardware when deciding how to apply their network policies. The quirks of these devices is evidence of the need to better understand the behaviour of SDN switches. As shown in the thesis, a better understanding can be achieved just through comparison with analytical results.

## 8.2 Future Work

This section presents the next steps for the results of the work in this thesis.

One purpose of this thesis is to provide guidance to future queueing models. This would lead to incorporating the effect of the switch's interface component into future models. Also investigating the effect of this in software switches.

As stated in this thesis and other literature there are multiple service delays seen across the switch. They are caused by the way the CPU of the switch interacts with the data path. An investigation into other ways this interaction can cause delays would be useful for performance evaluation in the future.

The work in this thesis leads to the question of whether other SDN switches experience the same behaviour. The experimental results of other models would benefit the results presented here.

An exclusively proactive network scenario is not something which benefits from modelling of both the control and data plane traffic together. However some results hint that there might be interference with the data path when flow rules are updated. Future analytical work might investigate and analyse this effect.

# Bibliography


[1] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)* (July 2008), 1–269.

[2] ARSHADI, L., AND JAHANGIR, A. H. On the TCP Flow Inter-arrival Times Distribution. In *UKSim European Symposium on Computer Modeling and Simulation (EMS)* (Madrid, Nov 2011), pp. 360–365.

[3] AZODOLMOLKY, S., NEJABATI, R., PAZOUKI, M., WIEDER, P., YAHYAPOUR, R., AND SIMEONIDOU, D. An Analytical Model for Software Defined Networking: A Network Calculus-based Approach. In *IEEE GLOBECOM* (Atlanta, GA, Dec 2013), pp. 1397–1402.

[4] AZODOLMOLKY, S., WIEDER, P., AND YAHYAPOUR, R. Performance Evaluation of a Scalable Software-defined Networking Deployment. In *Proceedings of European Workshop on Software Defined Networks (EWSDN)* (Berlin, 2013), IEEE, pp. 68–74.

[5] BIFULCO, R., AND MATSIUK, A. Towards Scalable SDN Switches: Enabling Faster Flow Table Entries Installation. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), SIGCOMM, ACM, pp. 343–344.

[6] GOTO, Y., MASUYAMA, H., NG, B., SEAH, W. K. G., AND TAKAHASHI, Y. Queueing Analysis of Software Defined Network with Realistic OpenFlow-Based Switch Model. In *IEEE 24th International Sym-*

*posium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)* (Sept 2016), pp. 301–306.

[7] HU, J., LIN, C., LI, X., AND HUANG, J. Scalability of control planes for Software defined networks: Modeling and evaluation. In *IEEE International Symposium of Quality of Service (IWQoS)* (Hong Kong, May 2014), pp. 147–152.

[8] HUANG, D. Y., YOCUM, K., AND SNOEREN, A. C. High-Fidelity Switch Models for Software-Defined Network Emulation. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking* (New York, NY, USA, 2013), HotSDN '13, ACM, pp. 43–48.

[9] HUANG, J., HE, Y., DUAN, Q., YANG, Q., AND WANG, W. Admission Control with Flow Aggregation for QoS Provisioning in Software-defined Network. In *IEEE GLOBECOM* (Austin, TX, Dec 2014), pp. 1182–1186.

[10] HUEBNER, F., LIU, D., AND FERNANDEZ, J. M. Queueing Performance Comparison of Traffic Models for Internet Traffic. In *IEEE GLOBECOM* (Sydney, NSW, 1998), pp. 471–476.

[11] JARSCHEL, M. *An Assessment of Applications and Performance Analysis of Software Defined Networking*. Doctoral thesis, Institute for Communication, Julius Maximilians University, Würzburg, 2014.

[12] JARSCHEL, M., OECHSNER, S., SCHLOSSER, D., PRIES, R., GOLL, S., AND TRAN-GIA, P. Modeling and Performance Evaluation of an OpenFlow Architecture. In *Proceedings of IEEE International Teletraffic Congress (ITC)* (San Francisco, CA, 2011), pp. 1–7.

[13] KREUTZ, D., RAMOS, F., ESTEVES VERISSIMO, P., ESTEVE ROTHENBERG, C., AZODOLMOLKY, S., AND UHLIG, S. Software-Defined Net-

working: A Comprehensive Survey. *Proceedings of the IEEE 103*, 1 (Jan 2015), 14–76.

[14] MAHMOOD, K., CHILWAN, A., ØSTERBØ, O. N., AND JARSCHEL, M. On the Modeling of OpenFlow-Based SDNs: The Single Node Case. In *Proceedings of the Sixth International Conference on Networks & Communications (NeCoM)* (Dubai, UAE, 7-8 Nov 2014).

[15] MAHMOOD, K., CHILWAN, A., ØSTERBØ, O. N., AND JARSCHEL, M. Modelling of OpenFlow-Based Software-Defined Networks: The Multiple Node Case. *Networks, IET 4*, 5 (2015), 278–284.

[16] OSGOUEI, A. G., KOOHANESTANI, A. K., SAIDI, H., AND FANIAN, A. Analytical Performance Model of Virtualized SDNs Using Network Calculus. In *Proceedings of Iranian Conference on Electrical Engineering (ICEE)* (Tehran, May 2015), pp. 770–774.

[17] PAXSON, V., AND FLOYD, S. Wide Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking 3*, 3 (Jun 1995), 226–244.

[18] PHEMIUS, K., AND THALES, M. B. OpenFlow: Why Latency Does Matter. In *IFIP/IEEE Symposium on Integrated Network Management* (Ghent, May 2013), pp. 680–683.

[19] TURULL, D., HIDELL, M., AND SJDIN, P. Performance Evaluation of OpenFow Controllers for Network Virtualization. In *High Performance Switching and Routing (HPSR)* (July 2014), pp. 50–56.