A Stochastic Model for Transit Latency in OpenFlow SDNs

Uzzam Javed[a], Azeem Iqbal[a], Saad Saleh[a], Syed Ali Haider[b], Muhammad U. Ilyas[b,*]

[a] *Applied Network & Data Science Research Group (AN-DASH)*
School of Electrical Engineering & Computer Science (SEECS)
National University of Sciences & Technology (NUST)
Islamabad — 44000, Pakistan
[b] *Department of Computer Science*
University of Jeddah, Jeddah, Saudi Arabia

Abstract

Software defined networks (SDNs) introduced the concept of decoupling control and data planes which is a paradigm shift. The OpenFlow protocol is one of a number of technologies that enables this decoupling and, in effect, commodifies network equipment. As of now, there is still limited work that has been done towards modeling the transit delay across OpenFlow switches experienced by network traffic. In this work we develop a stochastic model for the path latency in Open vSwitch (used together with a POX controller) based on measurements made in experiments performed on three different platforms which include 1) Mininet, 2) MikroTik RouterBoard 750GL and 3) GENI testbed softswitch. We propose a log-normal mix model (LNMM) and show that it offers a R^2 value of greater than 0.90 for most of our experiments. We also demonstrate how the M/M/1 models proposed in earlier studies is a poor fit.

Keywords: Software defined networks; OpenFlow, Open vSwitch; latency; modeling.

1. Introduction

The rapid evolution of new technologies, such as cloud computing, have complicated the way in which traditional networking was done [1]. Software defined networking (SDN), a new networking paradigm, provides a solution for it by smartly managing and configuring the network elements [2]. It makes the network programmable by separating the control plane of the network from the

*Corresponding author
Email addresses: 13mseeujaved@seecs.edu.pk (Uzzam Javed), 13mseeaiqbal@seecs.edu.pk (Azeem Iqbal), saad.saleh@seecs.edu.pk (Saad Saleh), usman@ieee.org (Muhammad U. Ilyas), ali.haider@seecs.edu.pk (Syed Ali Haider)

data plane. The data plane comprises of only switches with the capability of forwarding packets according to simple instructions received from the control plane. SDN provides the network administrators more control over the entire network through the centralized control plane running in software. It has also allowed researchers to experiment on deployed networks without causing any interference to their traffic [3].

The centralized control plane consists of a southbound application programming interface (API), for communication with the networks hardware comprising the data plane, and a northbound API, for communication with network applications and creation of networking functions [4]. OpenFlow is the principal southbound API, promoted by the Open Networking Foundation (ONF). The goal of OpenFlow is to provide open interoperability between network equipment of different vendors, as previous attempts made to this end were proprietary solutions. This effort improves the performance of SDNs [1].

The OpenFlow protocol helps to manage various elements of a network, *e.g.* implementing configuration changes in the data plane of a complex network such as a data center or telco's core network. However, these processes must be completed in timely manner. In traditional networks the control plane resides on each individual switch in a fragmented / distributed fashion which rarely affects the performance of the data plane. In OpenFlow switches the delay incurred by data plane elements (like switches and routers[1]) to process packets increases due to the involvement of a central controller. The increase in latency is due to: (1) the propagation delay between OpenFlow switch and control plane, (2) the processing speed of the control plane, and (3) the responsiveness of OpenFlow switches in generating flag for a new flow and updating their respective flow tables on receiving signal from the central controller [5]. Collectively, the sum total of these delays incurred by a packet at a switch is the store-and-forward delay. The goal of this paper is to measure and analyze the characteristics of all the delay components in the path of OpenFlow switches.

We developed a stochastic model based on measurements taken from three different OpenFlow switch platforms which include Mininet emulator, a MikroTik 750GL router and a production-level Open vSwitch (OVS) softswitch in the GENI testbed. These models will help us better understand the delay characteristics of Internet traffic in networks using OpenFlow controlled switches. Furthermore, we included Mininet, which is a virtual network emulator, in this measurement study in order to assess its accuracy in terms of delay. The stochastic model will help network designers and administrators anticipate expected end-to-end delays in WANs, overlay WANs and Internet links built from OpenFlow switches.

The rest of this paper is organized as follows. Section 2 contains information regarding work related to our problem, Section 3 formally defines our problem statement, Section 4 discusses our experimental setups, 5 presents the results

[1] We will use the term *switch* to collectively refer to L3 switches and routers.

through the experiments, Section 6 presents a stochastic model based on our empirical data. Finally Section 7 concludes the report by summarizing our results, drawing conclusions and giving possible directions for future research.

2. Related Work

The number of studies on the effects of adopting centralized control planes in SDN architectures on the delay performance are few. Bianco *et al.* [3] measured throughput and latency of an OpenFlow 0.8.9r2 softswitch built on an Ubuntu PC that performed L2 Ethernet switching using a Linux bridge (`bridge-utils`), L3 IP routing using Linux `ip_forward`, and an OpenFlow controlled virtual switch for different packet sizes and load conditions. He *et al.* [5] performed experiments on four types of hardware SDN switches to measure the latency involved in generation and execution of control messages. They focused their attention on the insertion delay and the effect of the position number of that rule has on it. Huang, Yocum and Snoeren [6] measured and compared the latency across three different hardware switches and OVS, and built an OVS based emulator for the physical switches whose latency closely mimics that of a particular physical switch. Sünnen [7] compared the delay performance of legacy switches to an NEC OpenFlow switch. Levin *et al.* [8] compared the performance of centralized control plane to that of a distributed control plane. This study was motivated by the fact that a centralized control plane cannot provide the same reliability and scalability as a distributed control plane. Heller, Sherwood and McKeown [9] also considered scalable and fault-tolerant OpenFlow control plane by adding more than one controller.

There has also been some work on the development of performance analysis benchmark suites for the control and data planes of SDNs. Among them, OFLOPS [10] is an open framework for the performance measurement of OpenFlow-enabled switches, while Cbench [11] is used to benchmark controller performance. These studies evaluated performance of OpenFlow architecture based on experimentation on hardware or simulations, on different OpenFlow platforms.

However, to the best of our knowledge, Jarschel *et al.* [12], Chilwan *et al.* [13], Yen *et al.* [14], Azodolmolky *et al.* [15], Bozakov *et al.* [16] and Samavati *et al.* [17] are the only studies to date to have developed delay models for OpenFlow networks. Jarschel *et al.* [12], Chilwan *et al.* [13] and Yen *et al.* [14] developed delay models based on queuing theory. However, Azodolmolky *et al.* [15] and Bozakov *et al.* [16] used network calculus to develop delay models.

Both approaches made some simplifying assumptions for analysis. Jarschel *et al.* [12], Chilwan *et al.* [13] and Yen *et al.* [14] assumed Poisson packet arrivals, whereas several studies have demonstrated that Ethernet traffic is self-similar (fractal) in nature, and is not accurately modeled as a Poisson process [18]. On the other hand, network calculus is a relatively new alternative to classical queueing theory. It has two branches: Deterministic network calculus (DNC) and stochastic network calculus (SNC). DNC, used by both Azodolmolky *et al.*

[15] and Bozakov *et al.* [16], only provides worst-case bounds on performance metrics and yields result that are of little practical use [19].

Samavati *et al.* [17] provided the comparison of performance in OpenFlow network of different topologies. They used graph theory for evaluation of this performance but did not consider controller-switch interactions in any significant detail.

3. Problem Formulation

The SDN revolution relies on four innovations; the separation of control and data planes, centralization of control plane, programmability of the control plane and standardization of application programming interfaces (APIs) [4]. Figure 1 shows all these features of SDN. The centralization of the control plane makes it easier to manage control changes within a network and makes much faster decisions than distributed protocols, by sensing and adjusting to state changes. The northbound API of the control plane is a key feature by which a large number of networking hardware can be orchestrated simultaneously. It allows a network to be divided into virtual networks having distinct policies, and yet reside on the same hardware. The control plane architecture consists of multicore support, switch partitioning and batch processing features for multiple input applications. Either incoming switch connections are distributed among worker threads or packets are stored in a common queue from which worker threads dequeue packets for processing [20]. A possible solution to the scalability challenges of having a centralized control plane, is the division of the network into smaller subnets with their own controllers.

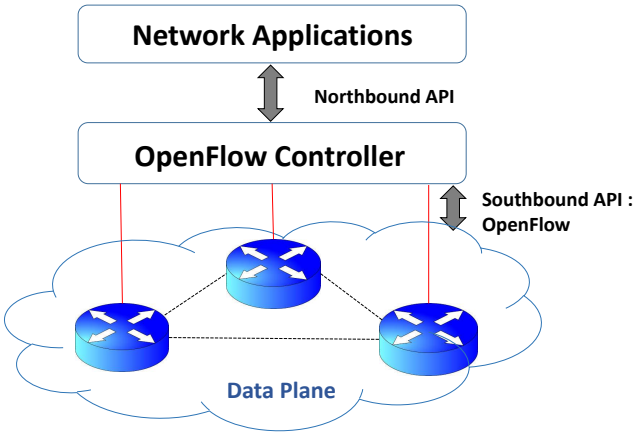


Figure 1: SDN Architecture.

SDN switches typically consult controllers for the first packet of every new flow. Subsequent packets of the same flow are processed according to the flowtable entry installed in response to the first packet. The traffic between

controller and forwarding elements is then significantly reduced by flow based traffic. However, this behavior is ultimately up to the way the network application running on the controller is designed.

The packet processing pipeline of an OpenFlow switch consists of three elements: (1) A flow table for processing and forwarding a packet, (2) a secure channel that connects a switch and the controller, and (3) the OpenFlow protocol, an open standard for communication between controller and switch. OpenFlow defines several different types of messages that can be exchanged between switch and controller [21]. The involvement of the controller by an OpenFlow switch in making forwarding decisions for some arriving packets means there can be a significant increase in processing delay. Figure 2 depicts the sequence of steps by which an OpenFlow switch processes a packet.

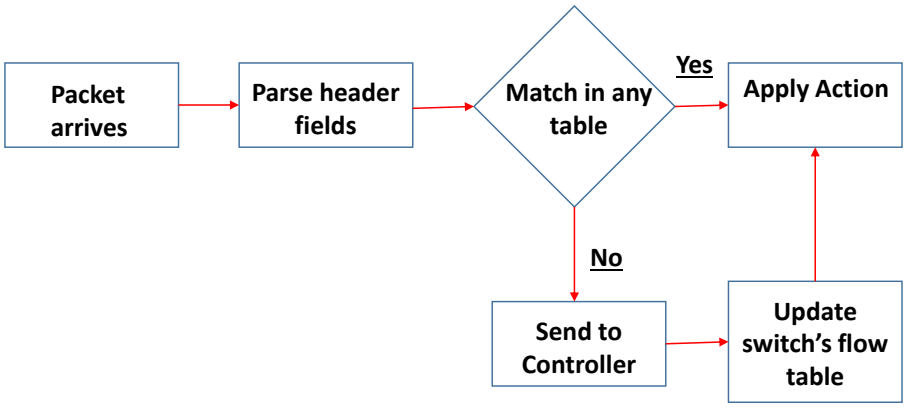


Figure 2: Sequence of processing steps of packet passing through an OpenFlow switch.

When a packet arrives at an OpenFlow switch, it searches for a matching entry in its flow table. If a match is found the corresponding specified action is performed. Else, if no match is found, the packet is forwarded to the controller through a `packet_in` message. The controller handles the packet, usually by updating the switch's flowtable [22].

The delay added by a switch to a path is an important parameter of interest. In this study, we measure the delay in a number of OpenFlow switches for a variety of packet sizes, packet arrival rates and traffic ratios that do not require controller : require controller interaction. The OpenFlow switches considered include a MikroTik 750GL hardware switch and a production-level Open vSwitch soft-switch in the GENI testbed. We analyze the measurements and derive a stochastic model of processing latency in OpenFlow switches, which can be extended to different deployment scenarios. Furthermore, we performed the same measurements on an OpenFlow switch in the Mininet network emulator, and compared those measurements with those obtained from the hardware and soft-switches.

4. Experimental Setups

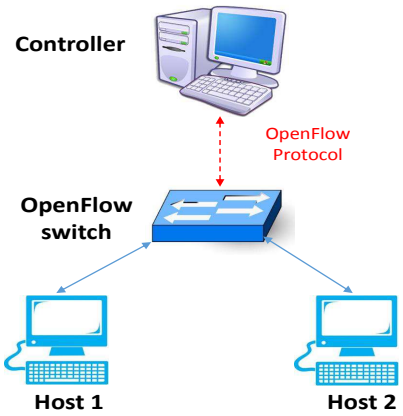


Figure 3: Experimental Setup.

To derive a stochastic model we first collected empirical data from experiments. For this purpose, we conducted experiments for three different OpenFlow switch platforms; which included Mininet, a MikroTik RouterBoard 750GL hardware Ethernet switch and a soft-switch in the GENI testbed. All switches used Open vSwitch (OVS), running OpenFlow 1.0, and were connected to a POX controller, to enable an unbiased comparison. Figure 3 depicts the basic experimental setup used to take measurements for all three platforms.

We used the *Distributed-Internet Traffic Generator* (D-ITG) [23] to generate and control repeatable network traffic flows for experiments. Using this traffic generator we measured the round-trip time (RTT) of sending a packet from one host/VM, through a switch to the other host/VM. This way we were able to use a single clock to measure packet RTT, and avoid clock synchronization issues we had to determine the time it takes a packet to travel one-way from sender to receiver from two different clocks. We considered synchronizing the clocks of sending and receiving hosts using the network time protocol (NTP) and measuring the time it takes for a packet to travel one-way from sender to receiver. However, NTP is only capable of providing clock synchronization up to millisecond resolution, which was insufficient for our experiments. Considering our experimental scenario, Figure 4 shows the total RTT latency of the experiment and its associated delay components.

According to Figure 4, four scenarios are possible while measuring RTT:

1. The switch does not consult the controller at all, as all the required flows are present in switch's flow table.
2. The controller is consulted by the switch on the forward path for the round trip flow.

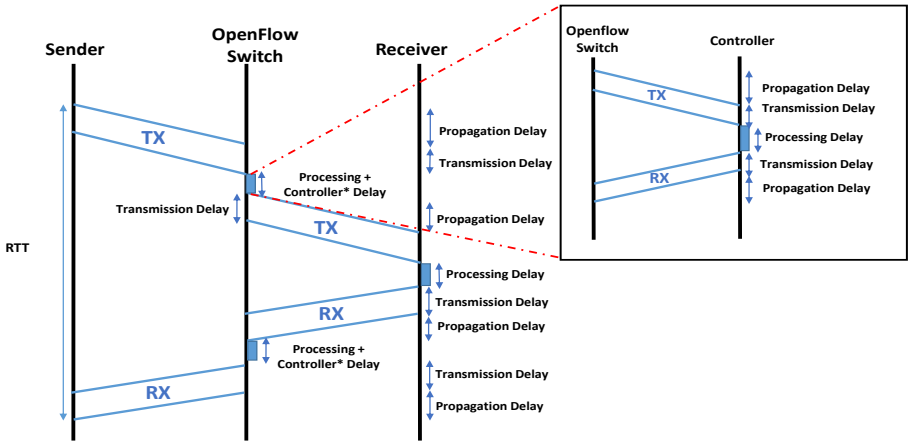


Figure 4: Latency per experiment. * Delay associated with controller may not be present all the time.

3. The switch does not consult the controller during the forward path, but consults it on the reverse path as the flow table entry has expired at that point in time.
4. The switch consults the controller both ways, on forward and reverse path, as a packet's flow table entry is not present at that point in time.

4.1. Mininet 2.2.1

Mininet [24] is a virtual network emulator widely used for prototyping software defined networks (SDN) on a single computer. We used Mininet 2.2.1, running OVS version 1.4.6, on a PC with Ubuntu Linux operating system (OS), Intel Core 2 Duo 3.0GHz processor and 2GBs of RAM. The minimal topology was used for the experiment.

4.2. MikroTik RouterBoard 750GL

For the MikroTik RouterBoard 750GL (RB750GL) hardware switch we replaced the default RouterOS it ships with, with OpenWRT Chaos Calmer (Bleeding Edge, r42655) release and ported OVS 2.3.0 onto it to enable it as an OpenFlow switch. The MikroTik RB750GL has 64MB RAM and a 64MB flash memory, which contains the full OS image. It has 5 wired Gigabit Ethernet ports and also supports VLAN configuration. Although the MitroTik RB750GL's accompanying documentation claims its factory installed RouterOS firmware supports OpenFlow 1.0 we found it to be unreliable and replaced it with the open source OpenWRT [25]. OpenWRT is essentially an embedded Linux distribution specifically designed for routers and switches. The router was connected with three PCs, all of same specifications which were 2GB RAM and Intel Core 2 Duo 3.0GHz processor and Ubuntu Linux OS installed on them. The network configuration file of the router was modified to make every port of

the router act as a separate interface. One PC was configured to host the POX controller and other two PCs served as sender (host 1) and receiver (host 2), respectively.

4.3. GENI Testbed Soft-switch

The Global Environment for Networking Innovation (GENI) [26] is a large-scale infrastructure experimentation testbed in network science. This testbed is overlaid over an already deployed backbone network, to provide access to existing network infrastructure without disrupting Internet traffic. We used it to measure the delay of OpenFlow switch at scale due to widely distributed resources in different locations. The setup was similar to the laboratory setup, but the resources were located across California, not in close proximity to each other. The OVS switch and the two hosts were located at CENIC InstaGENI aggregate located in La Mirada, California; while the chosen controller was located at InstaGENI aggregate in Stanford, California, roughly 450 miles away. The PCs had an Intel Xeon 2.10GHz processor with 512MB RAM running Ubuntu Linux OS. The OVS soft-switch, version 2.3.1, was implemented on a host running Ubuntu Linux.

5. Experimental Results and Findings

Two scenarios were considered in experiments: *Reactive* and *proactive* forwarding, to depict real world traffic scenarios. The reactive forwarding mode is employed when a switch does not find a matching flow table entry for an incoming flow and consults the controller. The controller adds flow entries by sending a `flow_mod` event packet to the switch. We also measured the RTT latency for packets that require controller intervention by installing flow entries using the `packet_out` event, to resolve queries on a per-packet basis. On the other hand, the proactive forwarding mode is in effect when the controller populates the switch's flow table ahead of time. To measure delays in reactive forwarding, we installed rules for a pair of MAC addresses for a complete round-trip and set their timeout value to 1 second, which is the smallest possible value. This was done to maximize the number of reactively forwarded packet in one session. To analyze the delay associated with OpenFlow switches, we considered three IP packet sizes commonly observed in Internet traffic; 77bytes (small), 562bytes (medium) and 1,486bytes (large) [27]. All three sizes are less than TCP's maximum segment size (MSS) and Ethernet's maximum transmission unit (MTU) limit to avoid segmentation or fragmentation. The input traffic was transmitted at different constant rates in different experiments, to adequately visualize and characterize the latency distribution.

5.1. Performance Benchmarking

We began by performing basic performance benchmarking of the three OVS switch platforms that will guide us when selecting appropriate ranges of configuration parameters for subsequent experiments.

5.1.1. Packet Loss Rate

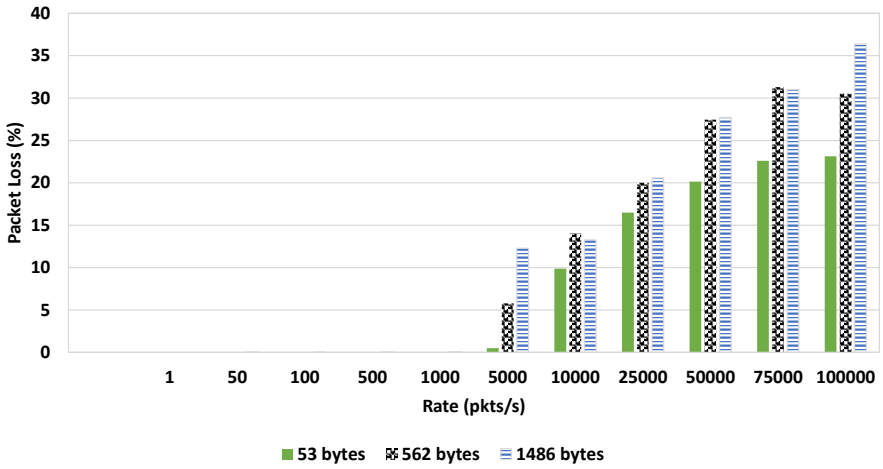


Figure 5: Packet loss rate of UDP traffic in MikroTik RB750GL at various packet transmission rates (5 minutes capture time per reading).

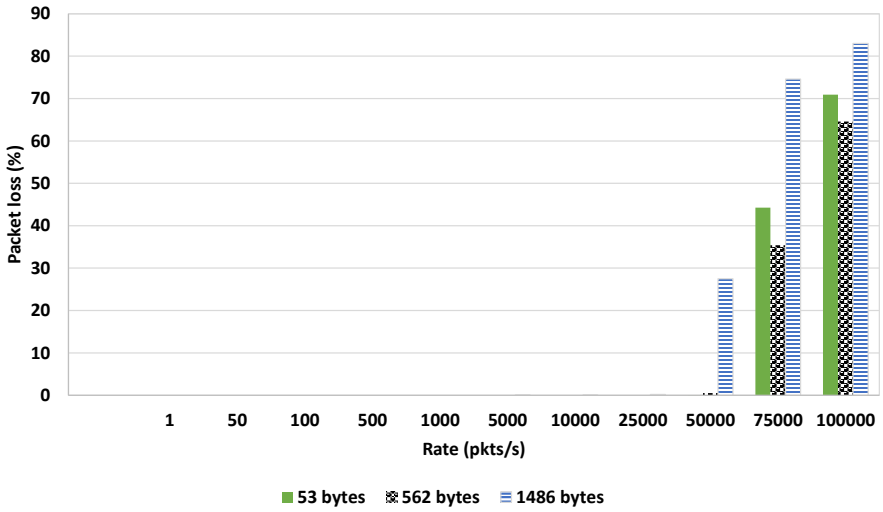


Figure 6: Packet loss rate of UDP traffic in GENI soft-switch at various packet transmission rates (5 minutes capture time per reading).

First, we began measuring the throughput capability of all three OVS switches by measuring the packet loss rate of UDP packets as a function of input packet rate. We define the packet loss rate as the fraction of IP+UDP packets sent by the sending host that are not received by the receiving host. We used D-ITG on

the transmitting host for the MikroTik RB750GL and GENI. The OVS switch in Mininet was operating under ideal conditions and would not show any packet losses at any packet transmission rate. For each packet transmission rate at the input to an OVS switch we measured packet losses over a period of 5 minutes. Figure 5 and Figure 6 show the trend in UDP packet loss rate for the MikroTik RB750GL and GENI OVS switches, respectively. Generally, packet loss rate grows with both packet transmission rate and packet size. We observed from Figure 5 and Figure 6, that below $1,000$pkts/sec there are little to no losses.

Similar to UDP loss rate, TCP retransmission rates were measured for MikroTik RB750GL and GENI, as shown in Figure 7 and Figure 8. We used the same experimental methodology as in the case of UDP loss rate. At the end of 5 minutes, the total number of packets retransmitted from host 1 were counted using the Wireshark file and then divided by the total time to find the packet retransmission rate. The trend is the same as for the UDP loss rate; overall packet retransmission rate increases with rate and packet size. There is a decrease in packet retransmission rate at high transmission rate of $10,000$ pkts/s due to the fact that the window capturing the effect closes after 5 minutes, because of which it is unable to capture all retransmitted packets. For the proactive forwarding mode we observed almost no losses.

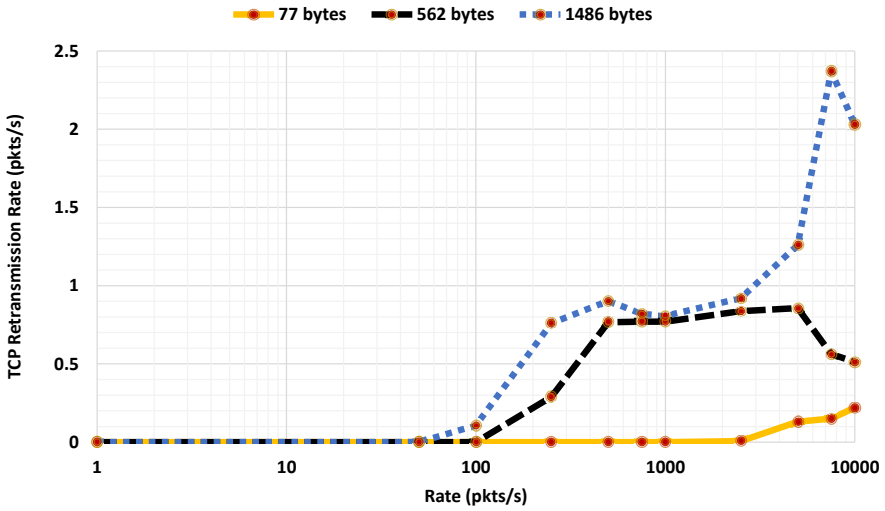


Figure 7: Retransmission rate in TCP traffic in MikroTik RB750GL (5 minutes capture time per reading).

5.1.2. Maximum Throughput Rate

We performed a series of experiments using *iperf* to measure the maximum possible throughput without losses. Table 1 tabulates the maximum bandwidth recorded in *iperf* experiments. It shows that there is a decreasing trend

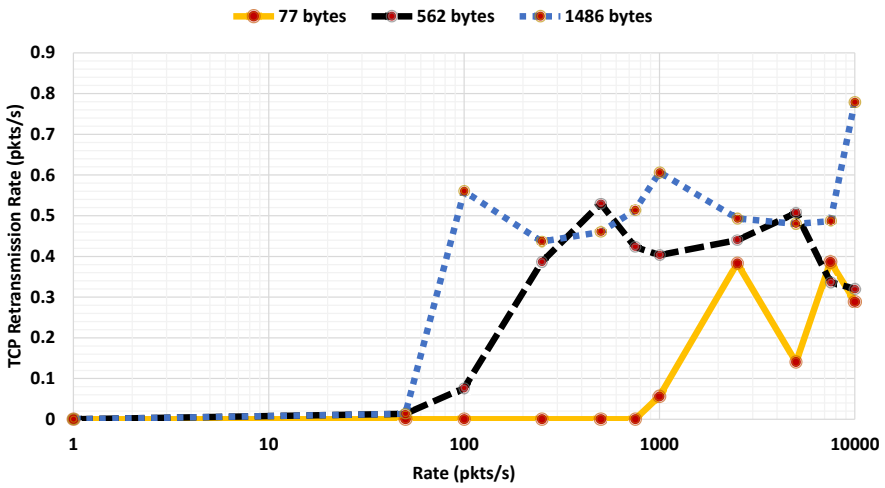


Figure 8: Retransmission rate in TCP traffic in GENI (5 minutes capture time per reading).

Table 1: Throughput comparison on different platforms using different scenarios (TCP window size = 85.3KBytes).

Platforms	Proactive	Reactive	Controller
Mininet	2.92Gbits/s	2.7Gbits/s	16.8Mbits/s
MikroTik RB750GL	514Mbits/s	477Mbits/s	6.29Mbits/s
GENI	99.5Mbits/s	83.6Mbits/s	6.08Mbits/s

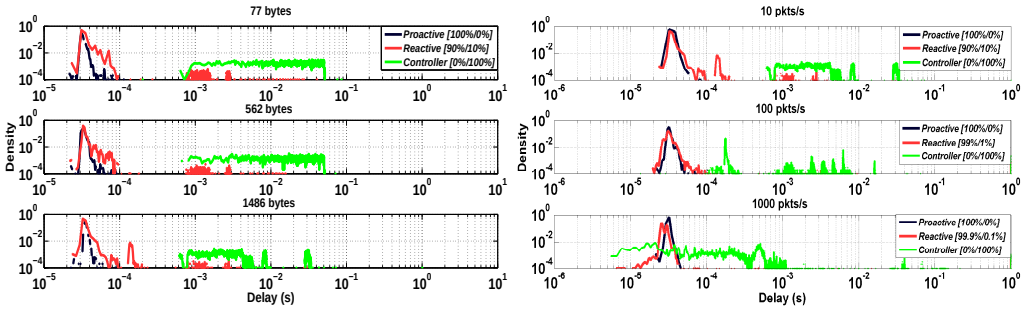
in throughput from proactive to reactive and then to controller. Across platforms, results show that the OVS switch in the Mininet emulator has the highest throughput, followed by the two physical switches, with GENI at the low-end.

5.1.3. Jitter

Table 2: Jitter comparison on different platforms using different scenarios (UDP buffer size = 208KBytes).

Platforms	Proactive	Reactive	Controller
Mininet	0.001ms	4.424ms	4.303ms
MikroTik RB750GL	0.011ms	1.449ms	1.766ms
GENI	0.113ms	5.225ms (1.3% loss)	3.027ms

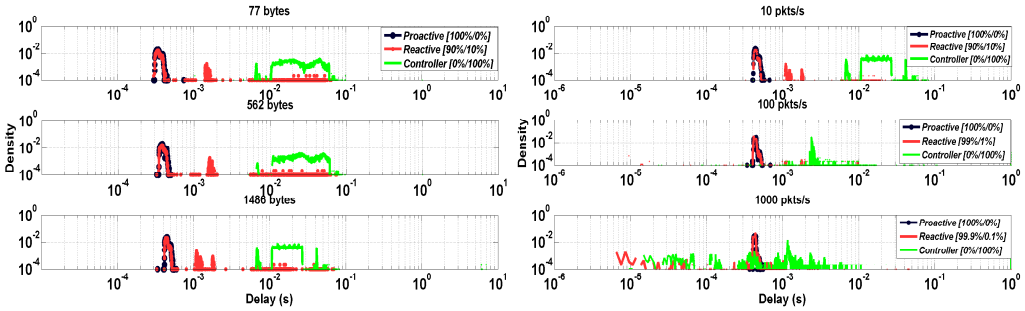
Similarly, Table 2 tabulates the jitter, which is the absolute difference between delay of two consecutive packets received of the same stream. It is measured across all three platforms under traffic scenarios that produce different ratios of proactively / reactively forwarded traffic. The results show that generally the delay jitter is highest in the reactive forwarding scenario,



(a) RTT latency vs. packet size (10 pkts/s input traffic rate).

(b) RTT latency vs. input traffic rate (packet size of 1,486 bytes).

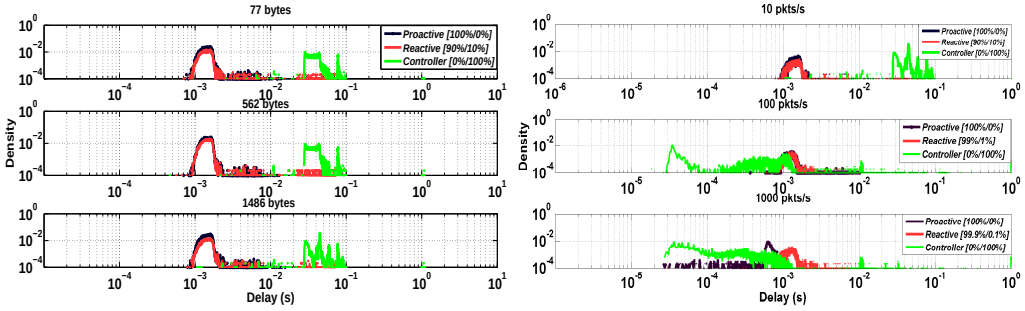
Figure 9: PDFs of RTT latency for Mininet.



(a) RTT latency vs. packet size (10 pkts/s input traffic rate).

(b) RTT latency vs. input traffic rate (packet size of 1,486 bytes).

Figure 10: PDFs of RTT latency for MikroTik RB750GL.



(a) RTT latency vs. packet size (10 pkts/s input traffic rate).

(b) RTT latency vs. input traffic rate (packet size of 1,486 bytes).

Figure 11: PDFs of RTT latency for GENI.

followed by the controller scenario and least for the proactive forwarding controller.

5.2. Latency vs. Packet Size

We began by measuring packet RTT latencies and plotting their Probability Density Functions (PDFs) for Mininet, MikroTik RB750GL and GENI OVS switches in Figure 9a, Figure 10a and Figure 11a, respectively, for three different packet sizes. For these measurements, we transmitted $10,000$ packets between two hosts at a constant rate of 10 packets per second (pkts/sec).This constant rate of input traffic was used for experiments to adequately characterize traffic distribution between switch and controller. For each of these three platforms we conducted the experiment once using only proactive forwarding (labeled proactive in figure legends), using a mix of proactive and reactive forwarding (labeled reactive in figure legends), and finally by consulting the controller before making a forwarding decision for every single packet (labeled controller in figure legends). In case of reactive forwarding mode, the timeout for a flow table entry was set to 1sec, the minimum possible. The format of the notation used in the legend of these figures denotes *[% of packets forwarded without controller intervention/% of packets forwarded with controller intervention]*. The variance of the RTT latency increases from proactive [100%/0%], to reactive [90%/10%] to controller [0%/100%] case. Figure 9a, Figure 10a and Figure 11a show that for all three OVS switches being tested, the packet size has no effect on the RTT latency, for all three packet forwarding modes. However, there is a significant difference in the order of magnitude of the RTT latencies seen on various platforms. The RTT latencies in Mininet were on the order of tens of microseconds, while for the MikroTik RB750GL they were on the order of hundreds of microseconds and for the GENI soft-switch on the order of milliseconds.

5.3. Latency vs. Packet Transmission Rate

In this series of experiments we investigated the effect of different packet transmission rates on the RTT latency. We varied the packet transmission rate from 10pkts/sec to 100pkts/sec to $1,000$pkts/sec for a stream of fixed packet size of $1,486$bytes. Figure 9b, Figure 10b and Figure 11b are PDF plots for Mininet, MikroTik RB750GL and GENI OVS switches, respectively. Note that because of the 1sec minimum timeout for flow table entries, when we use reactive forwarding the ratio of packets that get forwarded without controller intervention / packets that get forwarded with controller intervention changes with packet transmission rate. For example, at the packet transmission rate of 10pkts/sec the ratio is 90%/10%. Similarly, at the packet transmission rate of 100pkts/sec the ratio is 99%/1%, and at the packet transmission rate of $1,000$pkts/sec the ratio becomes 99.9%/0.1%. As the input traffic rate increases and a smaller fraction of traffic requires controller intervention in making forwarding decisions, the latency PDF of reactive forwarding mode approaches that of the proactive mode. The greater frequency of incoming packet also means that there is lesser context switching between OVS and non-OVS processes. The more frequent use

of OVS also leads to more cache hits, which further reduces the cost of context switches.

5.4. Latency vs. Variable Transmission Rate

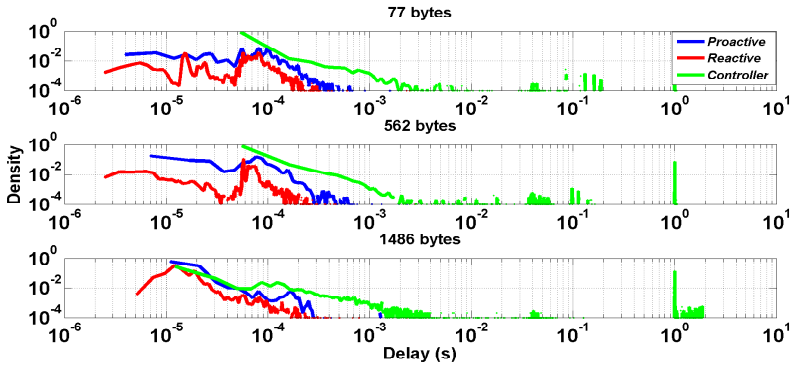


Figure 12: Delay versus packet size for Mininet at a variable Pareto input traffic rate.

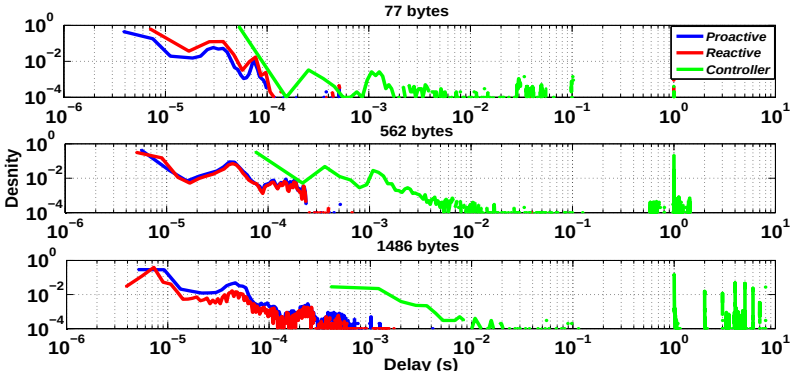


Figure 13: Delay versus packet size for MikroTik RB750GL at a variable Pareto input traffic rate.

The experiments we conducted and described this far were all based on constant bitrate traffic. In order to verify that the observations we made about latency distribution for constant bitrate traffic also extend to variable bitrate traffic we reran the same experiments on all three platforms. For these measurements, we transmitted $10,000$ packets between two hosts.

Figures 12, 13 and 14 are the latency distributions observed in Mininet, the MikroTik RB750GL and GENI OVS softswitch, respectively. Each figure covers the same three packet sizes of 77, 562 and $1,486$ bytes for each platform, like before. Garsva *et al.* [28] assumed variable packet transmission rate in their study and determined that inter-packet arrival times follow a Pareto distribution

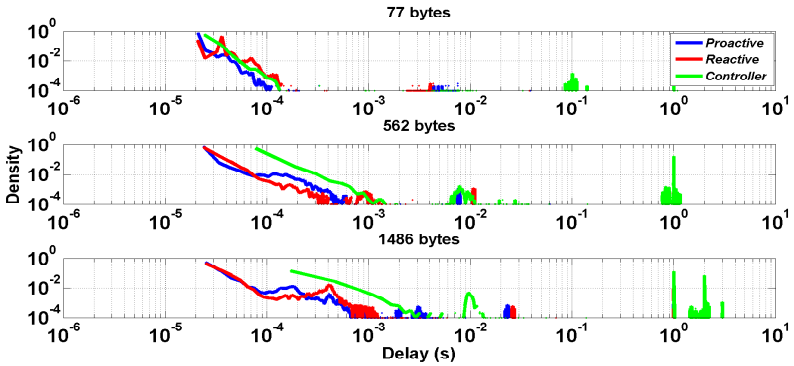


Figure 14: Delay versus packet size for GENI at a variable Pareto input traffic rate.

having shape and scale parameters 2.658 and 0.003, respectively. In the interest of using a realistic traffic scenario, we used the same parameters for the Pareto distribution for our traffic as reported by Garsva *et al.* in [28]. As Figures 12, 13 and 14 show, the latency distributions for all three platforms have bimodal distributions. The results of reactive forwarding largely agree with those of proactive forwarding as fewer packets consult the controller. The multi-modal PDF of controller is due to the queuing delay at the controller because of the high input traffic rate. These plots let us conclude that since the RTT latency PDF for variable input rate traffic remains similar to that observed for constant bitrate traffic, the model holds up for both cases.

5.5. Latency vs. Other Variables

We also measured latency by changing number of OpenFlow parameters which included the following:

1. Varying the number of fields against which to match a flow.
2. Varying the priority level of a matching flow table entry.
3. Changing the table in which the matching flow table entry is found.

However, none of these variations brought about measurable changes in the RTT latency. A likely explanation for this is OVS use of a tuple space search classifier, which creates a single hash table for same category of matches [29].

5.6. Observations

We made the following observations from the results of these experiments:

1. When the packet forwarding mode is exclusively proactive the distribution is unimodal, while in the case of mixed use of proactive and reactive forwarding the distribution is observed to be multimodal.
2. RTT latencies for mixed forwarding mode are concentrated around two clusters; the one on the lower end represents the latencies of packets forwarded proactively, while the one on the higher end represents the latencies of packets forwarded reactively and involved controller intervention.

3. The controller delay is about two orders of magnitudes larger than the processing delay at a switch.
4. There is an increase in latency with an increase in packet size in case of the MikroTik RB750GL router, while in the cases of Mininet and GENI there were no discernible changes in latency with packet size. This could be due to the fact that the overall propagation delay in setup is either small or large as compared to the individual delay components that is not recognizable.
5. The order of range of delay for packet sizes increases from Mininet, MikroTik RB750GL to GENI. A likely explanation for these differences are the differences in propagation delays between OpenFlow switch and POX controller across the three experimental setups.

6. Stochastic Modeling

Based on Figure 4 we model transit latency (L_T), the end-to-end delay in an OpenFlow SDN, as a sum of two components: Deterministic delay (D_D) and stochastic delay (D_S), i.e.

$$L_T = D + S. \tag{1}$$

These two terms are further decomposed in terms of the following equation:

$$L_T = (D_{trans} + D_{prop}) + (S_s + I \times S_c) \tag{2}$$

In Equation 2, D_{prop} is the propagation delay of a link to / from the switch, calculated as $D_{prop} = \frac{Distance}{Speed}$. D_{trans} is the transmission delay of a link to / from the switch, calculated as $D_{trans} = \frac{\text{Number\_of\_bits}}{\text{Link\_transmission\_rate}}$. This way, the deterministic delay D in Equation 1 is the sum of transmission delays D_{trans} and propagation delays D_{prop} of links on the path.

The terms S_s and S_c in Equation 2 are the stochastic delays associated with the switch and controller, respectively. I is a Bernoulli random variable that take on value 1 with probability α and value 0 with probability $(1 - \alpha)$, also called an *Indicator function*. The values of α depend on a variety of factors including the timeout value of flow table entries in switches and input traffic rate. The stochastic delay S in Equation 1 is the sum of all switch delays S_s and all controller delays S_c,

Due to the stochastic nature of the transit latency (L_T) we measured and modeled its PDF, as shown in Figures 9-11. We found the PDF of transit latency in OpenFlow switch SDNs to be multi-modal, which is a departure from the unimodal distributions of transit latencies found in traditional networks with distributed control planes. In Figures 9-11 plotted on log-log scales we can see that the PDFs exhibit multiple modes. We model these modes as log-normal

distributions[2]. We model the PDF as a log-normal mixture model (LNMM) of random variable X in Equation 3:

$$\begin{aligned} f_X(x) &= \sum_{i=1}^{K} \lambda_i \mathcal{N}(\log x; \mu_i, \sigma_i^2) \\ \text{where, } 0 \leq \lambda_i \leq 1 \text{ and } &\sum_{i=1}^{K} \lambda_i = 1, \end{aligned} \tag{3}$$

and $\mathcal{N}(x; \mu_i, \sigma_i^2)$ for $1 \leq i \leq K$ are K log-normal PDFs of the form,

$$\mathcal{N}(\log x; \mu, \sigma^2) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\left(-\frac{1}{2}\left(\frac{\log x - \mu}{\sigma}\right)^2\right)}. \tag{4}$$

In our case we set $K = 2$ to produce a bi-modal PDF of the log-normal mixture model. One mode represents the transit delay when the controller is consulted by the switch in making a forwarding decision and the other one for the case when the switch finds a matching flow-table entry. The parameters λ_1 and $\lambda_2 = 1 - \lambda_1$ are weights that determine the probability with which a packet is forwarded proactively (λ_1, without controller involvement) or reactively (λ_2, with controller involvement).

$$Mean = \mu e^{\frac{\sigma^2}{2}} \tag{5}$$

$$Variance = (e^{\sigma^2} - 1) e^{2\mu + \sigma^2} \tag{6}$$

The maximum likelihood estimates (MLEs) of the mean and variance parameters of a log-normal distribution using n samples x_i, where $1 \leq i \leq n$, are calculated as:

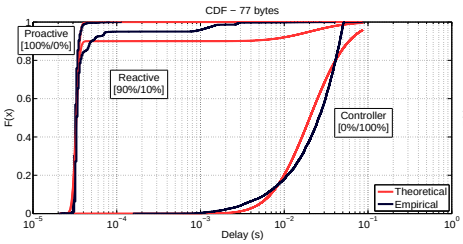
$$\overline{x} = \exp\left(\frac{1}{n}\sum_{i=1}^{n} \log(x_i)\right), \tag{7}$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \left(\log \frac{x_i}{\mu}\right)^2}. \tag{8}$$

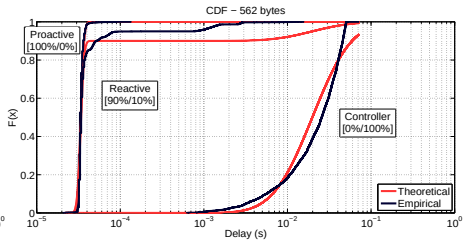
To validate the proposed stochastic model, we compared the model cumulative distribution function (CDF) against the empirical data for all platforms under considerations. We quantify the degree of similarity between model and data CDF using the coefficient of determination, denoted R^2, defined as,

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(x_i - f_X(x_i)\right)^2}{\sum_{i=1}^{n} \left(x_i - \overline{x}\right)^2}, \tag{9}$$

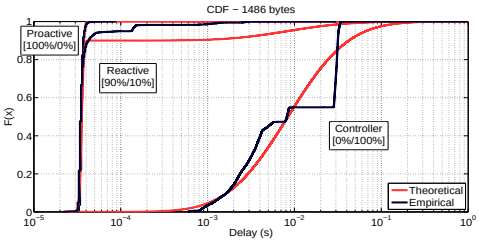
[2]A quantity's logarithm is distributed according to a Gaussian, then it follows the log-normal distribution [30]



(a) Proactive=0.97, Controller=0.92, Reactive=0.84

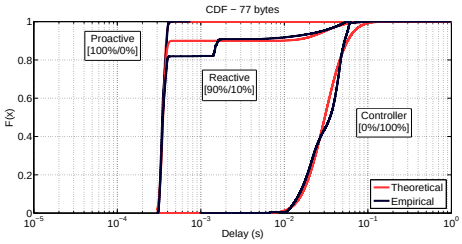


(b) Proactive=0.98, Controller=0.92, Reactive=0.90

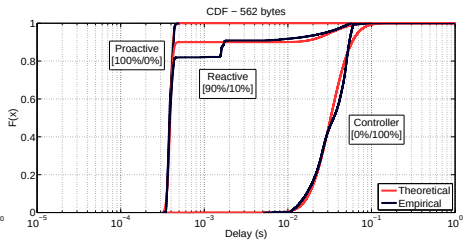


(c) Proactive=0.99, Controller=0.89, Reactive=0.90

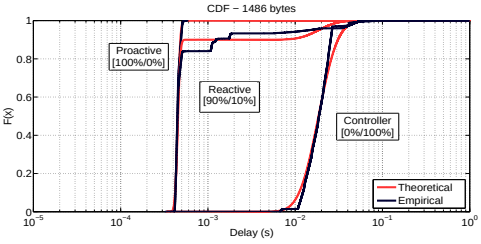
Figure 15: Model validation for Mininet, using CDFs for different scenarios and their respective R^2 values.



(a) Proactive=0.98, Controller=0.96, Reactive=0.98

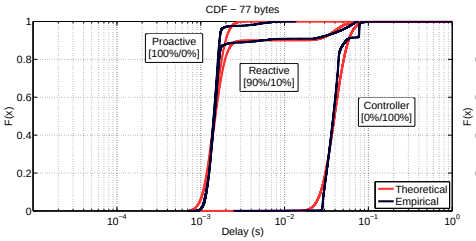


(b) Proactive=0.98, Controller=0.97, Reactive=0.98

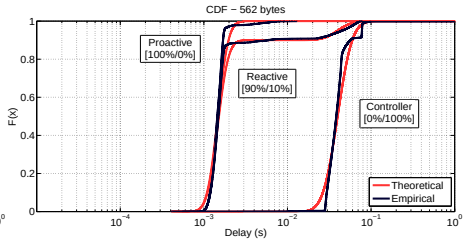


(c) Proactive=0.98, Controller=0.95, Reactive=0.98

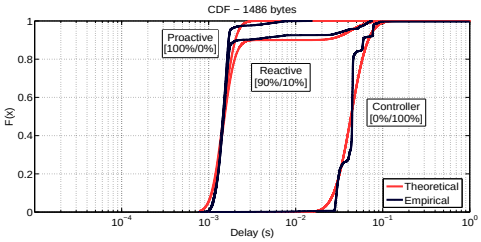
Figure 16: Model validation for MikroTik RB750GL, using CDFs for different scenarios and their respective R^2 values.



(a) Proactive=0.88, Controller=0.90, Reactive=0.94



(b) Proactive=0.90, Controller=0.89, Reactive=0.95



(c) Proactive=0.88, Controller=0.81, Reactive=0.92

Figure 17: Model validation for GENI, using CDFs for different scenarios and their respective R^2 values.

where $\overline{x}$ is the sample mean of the n samples.

Figure 15, Figure 16 and Figure 17 show various CDFs for Mininet, MikroTik RB750GL and GENI softswitch, respectively. R^2 can take on values in the interval $[0, 1]$, where values approaching 1 indicate a high degree of similarity between model and empirical data. Figure 18 shows the effect of having various various ratios of proactively / reactively forwarded traffic on the distribution of latency in Mininet OVS. It shows that for the proactive case [100%/0%] and the case where all packets go to the controller [0%/100%], the model produces unimodal distributions, and produces various bimodal distributions for various cases with different ratios for cases in between. Figure 15 plots the CDFs of the empirical data (plotted in black) against the modeled CDF $F_X(x)$ (plotted in red) for three different packet sizes (77, 562 and 1,486 bytes). Each subfigure contains three curves for three different traffic mixes; proactive [100%/0%], controller [0%/100%] and reactive [90%/10%]. Figure 16 and Figure 17 are similar plots for the MikroTik RB750GL and GENI softswitch platforms, respectively. We have given the R^2 value for each model vs. empirical distribution below each figure. The majority of R^2 values are above 0.90, with nearly half above 0.95. Even the lowest value is 0.81, which is still interpreted as a good fit. Therefore, generally the proposed log-normal mixed model is a good fit for the empirical data.

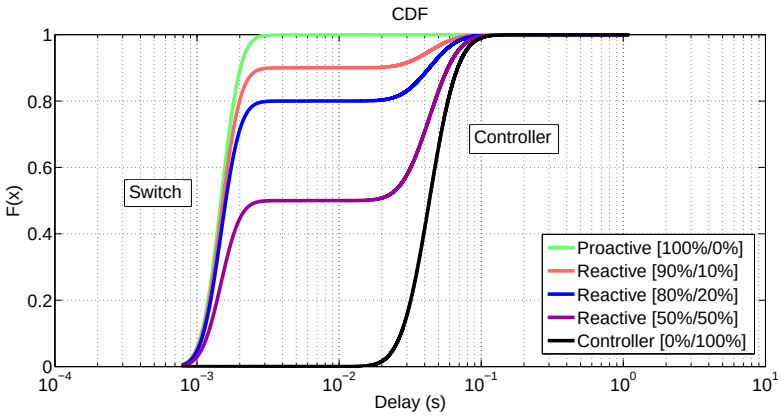


Figure 18: Effect of Switch and Controller interaction on overall distribution of Latency

7. Conclusions

In this paper, we presented a stochastic model for transit latency across OVS-based SDN switches in conjunction with a POX controller based on empirical data. We conducted numerous experiments, exploring the impact of a wide range of traffic parameters on three different OVS platforms: Mininet emulator, a MikroTik RB750GL router and GENI testbed. We proposed a log-normal mixture model distribution for transit latency and validated it with our experimental data and demonstrated it to be a good fit to empirical measurements. This model has the potential to be extended to a network having multiple switches and controllers. Previously developed models for latency across OpenFlow SDN switches derived from queueing theory are based on an M/M/1 model, i.e. assuming exponential packet inter-arrival and service times. Our results suggests that in terms of queuing models, an M/G/1 model, using a log-normal mixture model as the service distribution, would be more accurate.

References

[1] S. J. Vaughan-Nichols, Openflow: The next generation of the network?, Computer 44 (8) (2011) 13–15.

[2] H. Kim, N. Feamster, Improving network management with software defined networking, Communications Magazine, IEEE 51 (2) (2013) 114–119.

[3] A. Bianco, R. Birke, L. Giraudo, M. Palacin, OpenFlow switching: Data plane performance, in: Communications (ICC), 2010 IEEE International Conference on, IEEE, 2010, pp. 1–5.

[4] R. Jain, S. Paul, Network virtualization and software defined networking for cloud computing: a survey, Communications Magazine, IEEE 51 (11) (2013) 24–31.

[5] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, M. Thottan, Measuring control plane latency in sdn-enabled switches, in: Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, ACM, 2015, p. 25.

[6] D. Y. Huang, K. Yocum, A. C. Snoeren, High-fidelity switch models for software-defined network emulation, in: Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, ACM, 2013, pp. 43–48.

[7] D. Sünnen, Performance evaluation of openFlow switches, Ph.D. thesis, Eidgenössische Technische Hochschule (ETH), Zurich, Switzerland (2011).

[8] D. Levin, A. Wundsam, B. Heller, N. Handigol, A. Feldmann, Logically centralized?: State distribution trade-offs in software defined networks, in: Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks, ACM, 2012, pp. 1–6.

[9] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, in: Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks, ACM, 2012, pp. 7–12.

[10] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, A. W. Moore, Oflops: An open framework for openflow switch evaluation, in: Passive and Active Measurement, Springer, 2012, pp. 85–95.

[11] R. Sherwood, Y. Kok-Kiong, Cbench: an open-flow controller benchmarker (2010).

[12] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, P. Tran-Gia, Modeling and performance evaluation of an openflow architecture, in: Proceedings of the 23rd international teletraffic congress, International Teletraffic Congress, 2011, pp. 1–7.

[13] A. Chilwan, K. Mahmood, O. N. Østerbø, M. Jarschel, On modeling controller-Switch interaction in openFlow based SDNs, International Journal of Computer Networks & Communications 6 (6) (2014) 135.

[14] T.-C. Yen, C.-S. Su, An sdn-based cloud computing architecture and its mathematical model, in: Information Science, Electronics and Electrical Engineering (ISEEE), 2014 International Conference on, Vol. 3, IEEE, 2014, pp. 1728–1731.

[15] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, D. Simeonidou, An analytical model for software defined networking: A network calculus-based approach, in: Global Communications Conference (GLOBECOM), 2013 IEEE, IEEE, 2013, pp. 1397–1402.

[16] Z. Bozakov, A. Rizk, Taming sdn controllers in heterogeneous hardware environments, in: Software Defined Networks (EWSDN), 2013 Second European Workshop on, IEEE, 2013, pp. 50–55.

[17] M. Samavati, An analytical and performance study of random topologies in an openflow network.

[18] W. E. Leland, M. S. Taqqu, W. Willinger, D. V. Wilson, On the self-similar nature of ethernet traffic (extended version), Networking, IEEE/ACM Transactions on 2 (1) (1994) 1–15.

[19] F. Ciucu, J. Schmitt, Perspectives on network calculus: no free lunch, but still good value, in: Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication, ACM, 2012, pp. 311–322.

[20] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, S. A. Mehdi, An architectural evaluation of sdn controllers, in: 2013 IEEE International Conference on Communications (ICC), IEEE, 2013, pp. 3504–3508.

[21] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, ACM SIGCOMM Computer Communication Review 38 (2) (2008) 69–74.

[22] O. S. Consortium, et al., Openflow switch specification version 1.0. 0 (2009).

[23] A. Botta, A. Dainotti, A. Pescapè, A tool for the generation of realistic network workload for emerging networking scenarios, Computer Networks 56 (15) (2012) 3531–3547.

[24] B. Lantz, B. Heller, N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, ACM, 2010, p. 19.

[25] Enabling OpenFlow v1.0 on Mikrotik RouterBoard 750GL: A Tutorial, <`http://andash.seecs.nust.edu.pk/andash_publications/SDN.pdf`>.

[26] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, I. Seskar, GENI: A federated testbed for innovative network experiments, Elsevier Computer Networks 61 (2014) 5–23.

[27] R. Sinha, C. Papadopoulos, J. Heidemann, Internet packet size distributions: Some observations, USC/Information Sciences Institute, Tech. Rep. ISI-TR-2007-643.

[28] E. Garsva, N. Paulauskas, G. Grazulevicius, L. Gulbinovic, Packet interarrival time distribution in academic computer network, Elektronika ir Elektrotechnika 20 (3) (2014) 87–90.

[29] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, et al., The design and implementation of open vswitch, in: 12th USENIX Symposium on Networked Systems Design and Implementation, 2015.

[30] P. K. Janert, Data analysis with open source tools, O'Reilly, 2011, Ch. 9.