2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN) | 978-1-6654-3983-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/NFV-SDN53031.2021.9665124

Optimizing Flow Idle Timer in Reactive SDN

Gopinath Panda
Electrical & Computer Engineering
University of Central Florida, USA
Gopinath.Panda@ucf.edu

Shafaq Chaudhry
Office of Research
University of Central Florida, USA
Shafaq.Chaudhry@ucf.edu

Murat Yuksel
Electrical & Computer Engineering
University of Central Florida, USA
Murat.Yuksel@ucf.edu

***Abstract*—Reactive Software-Defined Networking (SDN) gives us more flexibility than traditional SDN by adding dynamic path definitions on top of data layer programmability driven by SDN controllers. This flexibility is necessary as it allows us to better utilize the limited capacity of flow tables that switches use to house the logic for handling traffic flows. However, it comes with a cost of control packets generated and exchanged between the controller and switch, processing time at the controller for the several types of control packets vying for its attention, and the waiting time at the switch as it awaits instructions by the controller. This may occur several times for a single traffic flow as the rules periodically expire due to each rule's idle timer and may need to be reinstalled into the flow table. This paper analyzes the delay incoming packets encounter in such a reactive SDN setup; presents a delay formulation based on modeling the system as M/M/1 queues at the controller and the switch; and studies the relationship between the idle timer, average delay, and flow arrival rate.**

***Index Terms*—Software-Defined Networking, reactive SDN, optimal idle timer, delay modeling.**

I. Introduction

Software-Defined Networking (SDN) [1] allows programmability of the control plane of the network that orchestrates the function of the data plane comprised of network devices like switches. Despite its origins from data center networking, SDN is now being considered as a mainstream networking paradigm with deployments spanning wide-area networks. One such protocol that enables SDN is OpenFlow [2], which allows information exchange between switches and controllers; as well as among controllers when multiple exist in the network. The controller can orchestrate globally optimal routes by installing these routes in various switches in response to the network state which it gathers by requesting various traffic statistics and network state information from network devices through OpenFlow. Thus, the network devices forward packets, but the logic of when these packets are forwarded and where to, or when they are dropped are directed by the controller. When this logic is pre-programmed, this is referred to as the proactive SDN approach; when the logic occurs in real-time, triggered by packet arrival events at the switch, this is referred to as a reactive SDN. Specifically, a switch learns the logic of packet handling by maintaining rules for different types of traffic flows in a structure known as the flow table. This flow table's entries, i.e., flow rules, are composed of header fields for matching the rule against incoming packets; an actions' list for handling the matching packet such as dropping it or forwarding it to a specific port; and a list of counters that keep track of various statistics such as how many packets have been matched against that flow rule.

In proactive SDN, flow rules are pre-configured into the switch flow table before any packets of flows are seen by the switch. These rules do not expire until explicitly removed. Packets arriving at a switch can be transmitted through at line rate, resulting in a predictable delay. Furthermore, if the switch loses its connection to the controller, it can continue to handle incoming traffic based on the rules. However, the switch has a limited memory resource. The flow table is typically implemented as a TCAM (ternary content-addressable memory) which has limited capacity as it is expensive, but it is very fast as it can search all its entries in a single clock cycle and can perform exact matches and pattern matches. According to [3], a 2-Mbit TCAM can hold around $6,200$ 15-tuple flow rules, each 356 bits in size. So, network administrators define broad-scoped rules using wildcards that match several different types of flows to one flow rule.

On the contrary, a reactive design enables efficient use of flow tables as rules can be installed for the duration needed. The rules are ephemeral and can expire due to inactivity, or maximum/hard timeout, or by an explicit flow rule removal command by the controller, or by the overflow policy configured at the switch. The first packet of a flow triggers flow setup delay due to interactivity between the controller and the switch. These include the time for the switch to look up a matching flow rule in its flow table. In case no flow rule is found, the switch needs to send an inquiry to the controller by means of a control message known as *Packet In*. This message may include part or all of the packet depending upon whether the switch has a buffer to store the packet while waiting on the response from the controller. The controller processes *Packet In* messages from various network devices in the order they are received. This is where the programmability aspect of SDN comes into play, where the controller centrally determines the optimal routes for this packet with a global view of the network. It will then generate a series of control messages to send to the switch that will result in installing a flow rule in the switch. This means there is a delay in processing *Packet In* messages at the switch impacted by how busy the controller is. There is also a network propagation delay experienced by the messages exchanged between the controller and the switch. Once the switch receives the controller messages to install flow rules, it may spend some time making space in the flow table to install

978-1-6654-3983-1/21/$31.00 ©2021 IEEE

the new rule(s) and then act upon the packet that was waiting in its buffer according to the action list of the installed rule.

The switch's reliance on the controller for instructions on handling incoming traffic flows opens the reactive setup to Denial-of-Service (DoS) attacks if the controller becomes overwhelmed [4]. This can happen if the controller is unable to process incoming messages in a timely fashion. If the flow rule idle timer value is set to expire prematurely, this may cause incoming packets of a flow to frequently generate *Packet In* messages to the controller – faster than the controller can process. If the number of control messages passed between the controller and the switch saturates the bandwidth between the controller and the switch, this can also choke the network leaving the switch in a waiting state for instructions. On the other hand, setting the idle timer too long loses the leverage a reactive SDN setup has over a proactive SDN in using a limited capacity flow table. Thus, setting the idle timer optimally is key to the proper function of a reactive SDN setup.

We dive deep into the mechanics of a switch and controller communication and derive the correlation of idle timer with the packet arrival rate, switch's packet processing time, and the controller's processing time of *Packet In* messages. Pointing out the limits of SDN will enable better handling of DoS attacks on an SDN setup. This study can further help network administrators in adjusting the idle timer values for different types of traffic flows, with different rates and Quality of Service requirements. Our contributions are summarized below:

- Delay analysis to show end-to-end latency an arbitrary packet experiences in a reactive SDN network that has a switch and a controller modeled as M/M/1 queues.
- The idea of a switch-controller interaction cycle to concentrate on the delay introduced as a result of flow management messages generated in a reactive SDN.
- For a single flow, we derive the delay formulation assuming an unbounded flow table, unbounded waiting queue at the switch and Poisson packet arrivals; and zoom in on the handling of packets at the switch leading up to flow rule installation and following after it as it processes any pending packets at its queue.

The rest of the paper is organized as follows: Section II describes the related work and inspiration for this work. Section III formulates the problem in light of some assumptions. Section IV presents delay analysis of the reactive SDN setup considering processing queues at both the controller and the switch; followed by the results presented in Section V. The concluding remarks and discussions follow in Section VI.

II. Related Work

Existing efforts are mostly empirical and do not provide insights into the limits of SDN. There are a few recent existing analytical models but they do not capture switch-controller interaction in detail and how this interaction impacts the end-to-end latency through a reactive SDN setup. It is important to understand this latency and the subsequent factors that play a role here in order to incur reduced packet delay while minimizing the cost of messages to the controller.

Several recent studies use queueing models to capture the behavior of an SDN setup by modeling the packet delay. In [5], the authors derive performance of the network by using an analytical model based on queueing theory. The packet sojourn time and packet loss were formulated by modeling a single SDN switch with unlimited queue space and a single SDN controller was modeled with limited buffer as an M/M/1-S queue. Poisson arrivals and departures were assumed with a single flow. In a similar vein, the work in [6] provided expressions for packet sojourn time and network throughput by modeling a single SDN node and single SDN controller as a Jackson Network, later extended to multiple switches attached to a single SDN controller and including the *Flow Modification* messages sent by the controller to the switch in the model [7]. Xiong et al. [8] derive a closed-form expression of the average packet sojourn time by modeling the controller and switch as M/G/1 and M^X/M/1. The authors in [9] model SDN-based edge cloud and present delay analysis of a packet passing through a switch at the edge cloud considering two 5G services: Enhanced mobile broadband and mobile Internet-of-Things. Graph-based models have been proposed to provide delay estimates such as those in [10]. Even though these studies are also using queueing models, they do not focus on optimizing the idle timer.

To reduce delay, several papers focus on improving switch flow table utilization, for example by setting flow rule idle timers intelligently. A dynamic idle timer and hard timeout adjustment algorithm is presented in [11] where the timer value is adjusted based on different traffic flows observing a 35% reduction of *Packet In* messages to the controller. The scheme presented in [3] adjusts the idle timer based on packet arrival intervals using two criteria, counting the number of mismatched flows, and the number of flows dropped by the controller. Intelligent eviction mechanisms can also be used to improve flow table utilization, such as the use of Machine Learning for classifying unused rules [12]. Although these studies focus on optimizing the idle timer like our work, they do not provide closed-form models and, hence, do not provide analytical insights to the performance limits emerging from idle timer configurations.

Studying delay and monitoring the flow table capacity is also necessary for the timely detection of DoS attacks where the flow table might get overwhelmed by unique packets being sent by an attacker to the switch generating a large number of rules that cannot fit into the limited TCAM [13]. Similarly, an attacker can generate packets at a rate that is just shy of the statically configured idle timer value, forcing the switch to react by sending messages to the controller causing controller to install rules for this fake traffic, thus starving legitimate traffic as well as causing delays in response time for legitimate packets [4]. Machine Learning approaches have been proposed in literature to detect such malicious attempts in SDN [14].

Our work presents a detailed analysis of the relationship between idle timer and average delay experienced by a packet taking into account the controller and switch communication signals. We show different cases of studying packet delay at

the switch: when a new packet first arrives; when additional packets arrive while switch is awaiting controller's instructions; and when switch encounters packets that match existing flow rules, but also has packets waiting to be processed in its queue. This work differs from the queueing model-based performance evaluation of SDN in [8] which focuses on modeling the controller, as our work dives deeper into what is going on within the switch. To the best of our knowledge, this is a unique attempt to derive a delay model that encompasses these various cases. We believe this foundational work can help network administrators define optimal idle timers for improving switch flow table utilization and can be used in the further study of improving security of SDNs.

III. Model Description

We start with modeling the simplest reactive SDN model with a single switch and single controller and a single flow from source to destination with packet inter-arrival times exponentially distributed. We use a queueing theory-based approach to model the reactive SDN and model both the switch and the controller as M/M/1 queues as illustrated in Figure 1. Packets are processed one by one at the switch. Packets that find no match in the flow table wait in the switch queue while the switch sends a *Packet In* message to the controller for further action. The controller processes incoming *Packet In* messages and forwards the flow installation or packet handling rules to the switch. Future *Packet In* messages from the switch wait in the controller queue for their turn.

Each flow table entry has two expiration timers corresponding to an idle timeout and a hard timeout, which are configured by the controller and govern when the rule is purged from the flow table. The idle timeout is the amount of inactivity time after which the flow rule is removed from the table. Here, inactivity refers to no incoming packets being matched against the flow rule which causes the idle timer to increment with each unit of time until it reaches the configured idle timeout value. When a packet is matched against a flow rule, its idle timeout is reset to 0. The hard timeout is the maximum time after which the flow rule is removed from the table regardless of activity. Both of these range from 0 through 65535 seconds.

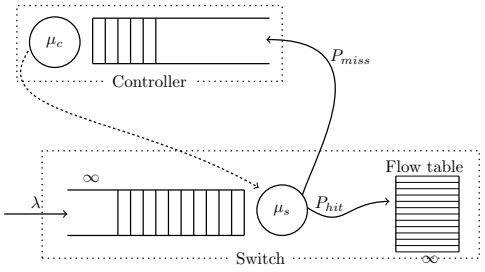

Fig. 1. Queueing model of a reactive SDN setup with a single flow.

We make the following simplifying assumptions:

- The switch queue has infinite capacity.
- Switch processes packets with Exponential rate μ_s.
- The flow table at the switch has infinite capacity.
- Controller processes *Packet In* messages at Exponential rate μ_c.
- Each flow rule is configured with an idle timeout, Δ.
- There is no hard timeout for the flow rules.

IV. Delay Analysis

We consider the case of a single flow arriving at the switch. Even though we assume infinite flow table capacity, derivation of the expected wait time of a packet requires careful attention due to multiple dynamic interactions in the system.

A. Miss Probability

Let A be the random variable that represents the generic inter-arrival times between packets of the flow. Here, A is Exponentially distributed with rate λ. A packet will match the entry in the flow table if $A \leq \Delta$; otherwise, it will miss if it arrives when the flow rule has expired and has been removed from the flow table. The packets that match a flow rule, i.e, hit the flow table, are processed according to the "action" field of the flow rule. Each missed packet waits in the switch queue and triggers the switch to send a *Packet In* message to the controller. Based on this switch behavior, we compute the steady-state probability of a miss (P_{miss}) as the joint probability that A is greater than Δ:

$$P_{miss} = \Pr(A > \Delta) = \int_{\Delta}^{\infty} \lambda e^{-\lambda x} dx = e^{-\lambda \Delta}. \quad (1)$$

Hence, the probability of a hit is:

$$P_{hit} = 1 - P_{miss} = 1 - e^{-\lambda \Delta}. \quad (2)$$

B. Missed Packets Before a Hit

For a new incoming flow, the first packet is always going to miss the flow table as there are no prior flow rules in the flow table. This causes a *Packet In* message to be sent to the controller to be processed with rate μ_c. While waiting for the flow rule to be installed, there will be an additional k missed packets. Let the delay in the installation of a new flow rule in the flow table be τ, which is the sum of the processing delay at the controller and the transmission delay from controller to the switch and the delay in installing flow rules received from the controller. Assuming the transmission delay, t_{cs}, to be a constant time unit, we have:

$$\tau = \frac{1}{\mu_c} + t_{cs}. \quad (3)$$

C. Switch-Controller Interaction (SCI) Cycle

Consider a time interval $[t_1, t_2]$ that consists of a Miss Phase (M) and a Hit Phase (H) as shown in Fig. 2. Specifically, the Miss Phase starts at time t_1 when the first packet of the flow arrives that does not find a match in the flow table. The Hit Phase starts when the flow rule is installed in the flow table until it is removed and ends by the next missed packet for this flow. This pattern emerges for a flow in a reactive SDN setup: First packet of a flow is a miss, then a sequence of misses occurs until the flow rule is installed by the controller, and a sequence of hits will follow. We define this combination

of Miss and Hit Phases as the *Switch-Controller Interaction (SCI) cycle* and base the rest of our analysis on this notion. The length of an SCI cycle depends on the idle timer and the packet inter-arrival times; it can be short (e.g., inter-arrival time of the packets is larger than the Δ) or infinitely long (e.g., Δ is longer than the longest possible packet inter-arrival time). Our model captures all the possibilities.

If we assume n packets in an SCI cycle, then there are $k+1$ misses and rest $n-k-1$ hits. Probability distribution of $k+1$ misses and $n-k-1$ hits also gives us the probability of having n packets in an SCI cycle $[t_1, t_2]$, which can be written as:

$$\pi_n = \sum_{k=0}^{n-1} P_{miss}^{k+1} P_{hit}^{n-k-1}. \tag{4}$$

The expected number of packets in an SCI cycle in the steady-state is:

$$\bar{n} = \sum_{j=1}^{\infty} j\pi_j = \sum_{j=1}^{\infty} j \sum_{k=0}^{j-1} P_{miss}^{k+1} P_{hit}^{j-k-1}. \tag{5}$$

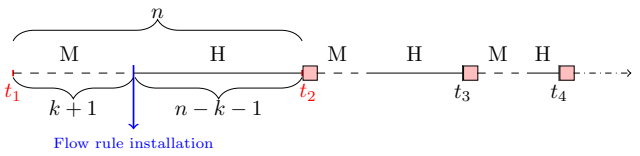

Fig. 2. Partition of time interval into SCI cycles with Miss and Hit phases.

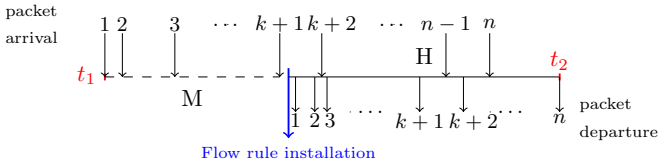

Fig. 3. Realization of packet arrivals and departures in an SCI cycle.

D. Expected Delay of a Packet

We now calculate the expected delay a packet experiences. Assume that packet i arrived; let's consider it the *tagged packet*. The tagged packet i can be any of the packets from 1 through n, and its delay can be computed under the following cases given that there are $k+1$ miss packets in the SCI cycle:
Case I: Tagged Packet Arrives During the Miss Phase. If $i \leq k+1$, the waiting packets are in a Miss Phase of the SCI cycle and the delay of the tagged packet is:

$$W_1(i) = \tau + \frac{i}{\mu_s}. \tag{6}$$

Case II: Tagged packet arrives during the Hit Phase. If $i > k+1$, the tagged packet must have arrived after the flow rule gets installed at the switch. This means the Hit Phase has started. There are two possibilities in this case:
Case II-A: Some of the Miss Packets are waiting to be processed. If $i > k+1$ and some waiting packets are in the Miss Phase of the SCI cycle and the rest in the Hit Phase, the delay of the tagged packet is going to be dependent on how many miss and hit packets are in the queue. If the tagged packet finds that $j < k+1$ of the miss packets are already processed, then $k+1-j$ miss packets are waiting in the switch queue of which $i-(k+1-j)$ are from the Hit Phase. Given that $j < k+1$ miss packets were served by the switch queue, the delay of the tagged packet is:

$$W_2(i,j) = \frac{i-j}{\mu_s}, \ i = 1, \ldots, n, \quad j = 1, \ldots, k. \tag{7}$$

Case II-B: All of the Miss Packets have been processed; and some of the Hit Packets may be waiting to be processed. If $i > k+1$ and all the waiting packets are from the Hit Phase of the SCI cycle, the delay of the tagged packet will be determined by M/M/1 delay expression:

$$W_3(i) = \frac{\lambda}{\mu_s - \lambda}. \tag{8}$$

The conditional delay of the tagged packet given that there are $k+1$ miss packets out of n packets in an SCI cycle is the sum of the delays in the three cases with their probability of occurrence. For Case-I, there must be i contiguous misses, with probability P_{miss}^{i}. For Case-II, there must be $k+1$ misses followed by $i-k-1$ hits where i can be from $k+2$ to n. This probability is expressed by $P_{miss}^{k+1} P_{hit}^{i-k-1}$. In Case II, two different delay expressions emerge depending on how many miss packets were processed before the arrival of the tagged packet. We can express the probability that j miss packets are processed before the arrival of the tagged packet as:

$$\Pr\{A > S_j\} = \int_0^{\infty} \frac{\mu_s^j x^{j-1}}{(j-1)!} \, e^{-\mu_s x} e^{-\lambda x} dx = \left(\frac{\mu_s}{\lambda + \mu_s}\right)^j, \tag{9}$$

where A is the inter-arrival time, and S_j is the processing time of j miss packets with probability density function:

$$f(x) = \frac{\mu_s^j x^{j-1}}{(j-1)!} \, e^{-\mu_s x}. \qquad j = 1, 2, \ldots \tag{10}$$

Bringing all the cases together, we express the average conditional delay of the tagged packet as:

$$W(n|(k+1)) = \sum_{i=1}^{k+1} P_{miss}^{i} W_1(i) + \sum_{i=k+2}^{n} P_{miss}^{k+1} P_{hit}^{i-k-1} \left[\sum_{j=1}^{k} \left(\frac{\mu_s}{\lambda+\mu_s}\right)^j W_2(i,j) + \left(\frac{\mu_s}{\lambda+\mu_s}\right)^{k+1} W_3(i) \right]. \tag{11}$$

Given that there are n packets in an SCI cycle, the total expected delay of a packet arriving during the SCI cycle is:

$$W(n) = \sum_{k=0}^{n-1} P\{n|(k+1)\} W(n|(k+1)), \ n \geq 2, \tag{12}$$

where $P\{n|(k+1)\} = P_{miss}^{k+1} P_{hit}^{n-k-1}$ is the probability that there are $k+1$ misses given n packets in the SCI cycle.

Finally, the total expected delay of a packet will be:

$$\bar{W} = \sum_{n=2}^{\infty} \sum_{k=0}^{n-1} P_{miss}^{k+1} P_{hit}^{n-k-1} W(n). \quad (13)$$

V. Numerical Results

In this section, we discuss the effects of system parameters on the average delay, $\bar{W}$, of a packet and explore the optimum settings of the idle timer, Δ. We consider the following parameters in the numerical experiment carried with Maple software: transmission delay between controller and switch, $t_{cs} = 0.5s, 1s, 1.5s$; idle timer, $\Delta = [5, 180]$s; packet arrival rate, $\lambda = [10, 300)$ packets/s; controller service time, $\mu_c = 200$ packets/s; and switch service time, $\mu_s = 300$ packets/s.

Fig. 4 depicts the effect of λ on $\bar{W}$ for different idle timers Δ. The average delay is a convex function of λ, with optimal values attained when $\lambda \in (200, 300)$. As expected, when idle timer values are larger, average delay decreases due to increased hit rate of incoming packets and subsequent reduced switch-to-controller communications. However, after a certain point, the average delay increases as packet arrival rates get close to the processing capacity of the system. This is observed in Fig. 4 when the packet rate is close to 300 packets/s.

For a given processing power of switch and controller, and transmission delay between switch and controller, there seems to exist an optimal packet arrival rate and idle timer value for which the average delay is minimized. The optimum arrival rates, λ^*, shown in Fig. 5, are essentially the minimum points of the delay curves in Fig. 4. Fig. 5 shows that the optimum traffic rate first decreases and then increases as the idle timer increases. When Δ is below 50s, the delay due to the need for the controller to re-install expired flow entry dominates the average delay the packets experience. So, the best arrival rate increases as Δ decreases. However, when the idle timer is long enough (i.e., Δ is above 50s), there is less need to worry about the controller's flow re-installation delay.

Fig. 5 also shows the effect of varying controller processing time on the optimum packet arrival rate. When $\Delta < 100$s, as controller processing rate is increased, a lower traffic rate is better for a faster controller processing rate. Even though a faster controller can reduce queueing delay at the controller, the main delay in the $\Delta < 100$s window is caused by the number of switch-controller interactions. Lower packet rates reduce the frequency of these interactions, so the combination of faster controller with slower traffic reduces both controller queueing delay as well as switch-controller interactions.

Fig. 6 shows that, in general, for a given packet arrival rate, λ, there is a value of idle timer, Δ, that is optimum, and corresponds to the minimum average delay. After this value, the average delay remains the same. This point reflects the case when every packet of flow finds a match in the flow table without requiring an installation of the flow rule. Specifically,

- Lower traffic rates ($\lambda \leq 250$) observe a reduction in delay as idle timer value increases but does not see as much reduction in delay as the higher traffic rates. This is because lower rates cause flow table misses and corresponding controller-switch overhead even with higher idle timer values.
- High traffic rates ($\lambda > 250$) reduce delay as idle timer value increases until it reaches the optimum minimum average delay after which an increase in the idle timer does not further decrease the delay as controller involvement is no longer needed (flow rule does not expire).
- At lower values of idle timer ($\Delta \leq 100$s), higher traffic rates ($\lambda > 250$) observe a higher delay than lower traffic rates ($\lambda \leq 250$). This is because of the queueing effect at the switch as more packets arrive and get queued after the first miss at faster traffic rates than slower traffic rates, while the switch awaits controller's instruction.

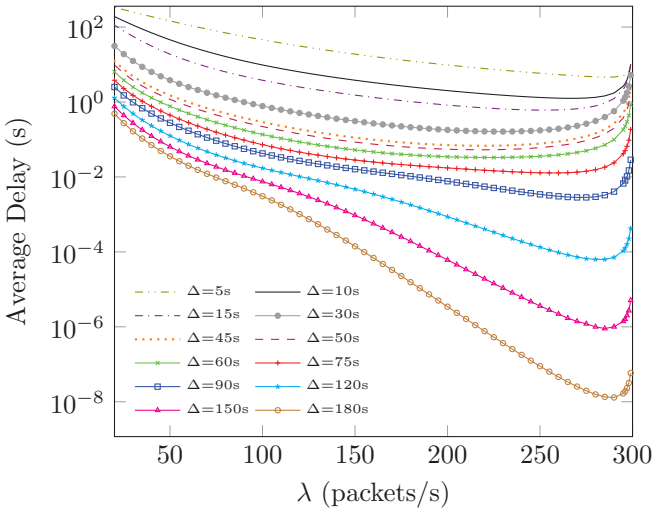

Fig. 4. Average packet delay vs. packet arrival rate for different idle timers ($\mu_s = 300, \mu_c = 200, t_{cs} = 1s$).

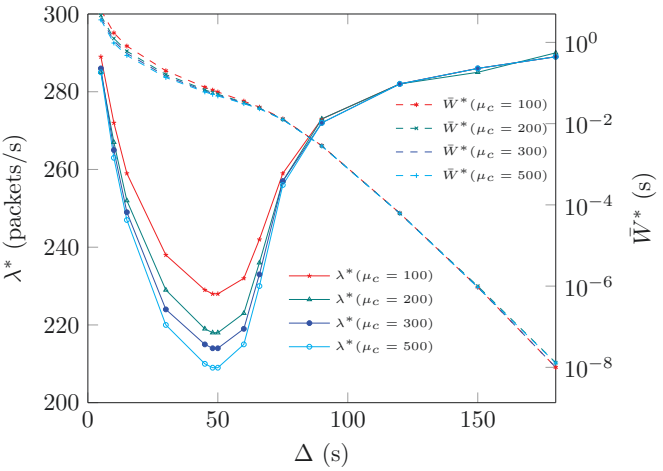

Fig. 5. Optimal packet arrival rate and optimal average packet delay for different idle timers ($\mu_s = 300, t_{cs} = 1s$).

VI. Discussion and Concluding Remarks

In conclusion, this paper presented a detailed analytical study of the average delay encountered by a packet that arrives

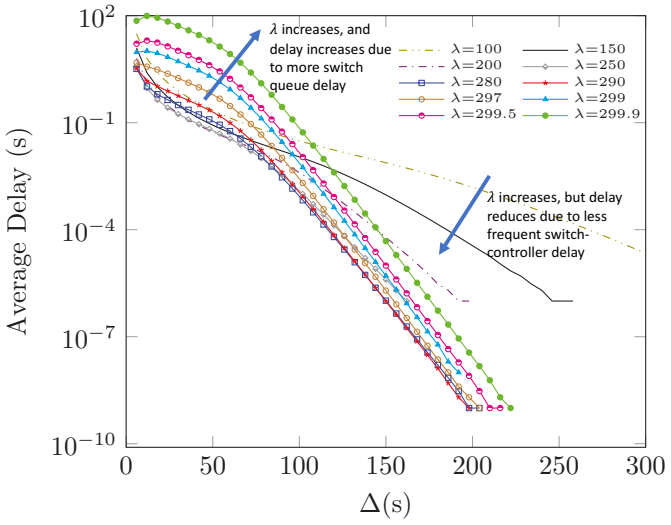

Fig. 6. Average packet delay vs. idle timer for different packet arrival rates ($\mu_s = 300, \mu_c = 200, t_{cs} = 1s$)

at a switch in a network with an SDN controller and switch. We show that the average delay decreases as the packet arrival rate increases before it increases again as the arrival rate exceeds system processing times. For different values of arrival rates, we also show the optimal values of idle timers that yield the minimum average delay.

In practicality, setting the idle timer must take into consideration other factors such as limited capacity of the switch's flow table which result in flow rule evictions regardless of matched activity. If flow rules expire too early, this incurs an overall cost in terms of the increased communication overhead between the switch and controller. If flow rules stick around too long, the flow table gets filled resulting in eviction overhead to make room for incoming flow rule logic by the controller. Thus, in addition to the rate of packet arrivals, the idle timer should be set based on flow table capacity as well as the eviction mechanism in place. The authors in [15] show that sometimes it may be beneficial to remove a flow rule before the flow ends to reduce average flow table occupancy.

In addition to the idle timer, Open Flow specification defines a hard timer that expires the flow rule regardless of matches. This ensures that flow rules get purged from the flow table periodically to continue making room for new rules. These two timers work together to govern when rules get removed from the flow table. The work in [16] shows that for a given eviction policy such as Least Recently Used, dynamically setting the hard timer performs better than a fixed hard timer.

Idle timer should also take into consideration the duration of flows and quality of service guarantees for different flow types, e.g., shorter flows with more real-time needs may need a larger idle timer to reduce the controller-switch communication overhead of re-installing the rule soon after expiration. This is usually accomplished by means of adaptive algorithms to set and adjust the idle timer values [17], [18].

In future work, we want to expand our analysis to consider flow table size, finite switch queue size, flow rule hard timer and multiple traffic flows with different traffic distributions. Additionally, we plan to study how well the analytics model compares using an emulation in Mininet.

Acknowledgment

This work was supported in part by U.S. National Science Foundation awards 1814086 and 1647189.

References

[1] P. Goransson, C. Black, and T. Culver, *Software Defined Networks, Second Edition: A Comprehensive Approach*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.

[2] "OpenFlow Protocol," https://www.opennetworking.org/technical-communities/areas/specification/open-datapath/.

[3] M. Lu, W. Deng, and Y. Shi, "TF-IdleTimeout: Improving efficiency of TCAM in SDN by dynamically adjusting flow entry lifecycle," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016, pp. 002 681–002 686.

[4] B. Isyaku, M. S. Mohd Zahid, M. Bte Kamat, K. Abu Bakar, and F. A. Ghaleb, "Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey," *Future Internet*, vol. 12, no. 9, 2020.

[5] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and Performance Evaluation of an OpenFlow Architecture," in *2011 23rd International Teletraffic Congress (ITC)*, 01 2011, pp. 1–7.

[6] A. Chilwan, K. Mahmood, O. N. sterb, and M. Jarschel, "On Modeling Controller-Switch Interaction in Openflow Based SDNs," *Int'l. Journal of Computer Networks & Communications*, vol. 6, pp. 137–150, 2014.

[7] K. Mahmood, A. Chilwan, O. Østerbø, and M. Jarschel, "Modelling of OpenFlow-based software-defined networks: the multiple node case," *IET Networks*, vol. 4, no. 5, pp. 278–284, 2015.

[8] B. Xiong, K. Yang, J. Zhao, W. Li, and K. Li, "Performance evaluation of OpenFlow-based software-defined networks based on queueing model," *Computer Networks*, vol. 102, pp. 172–185, 2016.

[9] A. Chilwan and Y. Jiang, "Modeling and Delay Analysis for SDN-based 5G Edge Clouds," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, 2020, pp. 1–7.

[10] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the Potential of Graph Neural Networks for Network Modeling and Optimization in SDN," in *Proceedings of ACM Symposium on SDN Research*, 2019, p. 140–151.

[11] B. Isyaku, M. B. Kamat, K. b. Abu Bakar, M. S. Mohd Zahid, and F. A. Ghaleb, "IHTA: Dynamic Idle-Hard Timeout Allocation Algorithm based OpenFlow Switch," in *2020 IEEE 10th Symposium on Computer Applications Industrial Electronics (ISCAIE)*, 2020, pp. 170–175.

[12] H. Yang, G. F. Riley, and D. M. Blough, "STEREOS: Smart Table EntRy Eviction for OpenFlow Switches," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 377–388, 2020.

[13] T. A. Pascoal, Y. G. Dantas, I. E. Fonseca, and V. Nigam, "Slow TCAM Exhaustion DDoS Attack," in *ICT Systems Security and Privacy Protection*, S. De Capitani di Vimercati and F. Martinelli, Eds. Cham: Springer International Publishing, 2017, pp. 17–31.

[14] S. Nanda, F. Zafari, C. DeCusatis, E. Wedaa, and B. Yang, "Predicting network attack patterns in SDN using machine learning approach," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016, pp. 167–172.

[15] S. Shirali-Shahreza and Y. Ganjali, "Delayed Installation and Expedited Eviction: An Alternative Approach to Reduce Flow Table Occupancy in SDN Switches," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1547–1561, 2018.

[16] A. Panda, S. S. Samal, A. K. Turuk, A. Panda, and V. C. Venkatesh, "Dynamic Hard Timeout based Flow Table Management in Openflow enabled SDN," in *International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, 2019, pp. 1–6.

[17] X. Xu, L. Hu, H. Lin, and Z. Fan, "An Adaptive Flow Table Adjustment Algorithm for SDN," in *Proceedings of IEEE HPCC/SmartCity/DSS*, 2019, pp. 1779–1784.

[18] Y. Liu, B. Tang, D. Yuan, J. Ran, and H. Hu, "A dynamic adaptive timeout approach for SDN switch," in *IEEE International Conference on Computer and Communications (ICCC)*, 2016, pp. 2577–2582.