qMon: A Method to Monitor Queueing Delay in OpenFlow Networks

Sandhya Rathee, Shubham Tiwari, K Haribabu, and Ashutosh Bhatia

***Abstract*—In software-defined networking (SDN), the decoupled architecture provides opportunities for efficiently measuring critical quality of service (QoS) parameters, such as delay. Existing approaches, to dynamically obtain delay, are based around calculating the transit time of a probe packet that travels through the data links. These approaches are not efficient as the probe packet injected into the data plane incurs considerable overhead. Additionally, a separate probe packet is required to measure the delay of each queue if more than one queue is present on the egress port of a switch. Thus, these approaches are not scalable. In this paper, we propose an efficient passive delay estimation method, queueing delay monitoring (qMon), to monitor queueing delay in SDN networks. qMon leverages the OpenFlow protocol to obtain queue statistics from switches at regular intervals, which are further employed to estimate the mean queueing delay for each interval. Thus, the proposed approach differs from the existing approaches as no packet is injected into the data plane to measure delay. The results show that for Poisson traffic and for bursty traffic with large ON intervals, round trip time (RTT) values estimated using qMon and ping utility demonstrate high correlation when the measured RTT value is considered as time-series data.**

***Index Terms*—OpenFlow, passive delay monitoring, queueing delay, software defined network.**

I. INTRODUCTION

OPENFLOW enables the separation of control plane and data plane in software-defined networking (SDN) by providing an interface to programmatically modify or retrieve the switch configuration. As such, it provides a holistic view of the network topology at the controller and enables fine-grained monitoring of the network links. A more precise view of the network provides opportunities for improving the efficiency of the quality of service (QoS) routing algorithms.

In recent years, many organisations have been encouraging their employees to use real-time applications such as remote access, voice and video conferencing to make the work environment flexible and increase the work output. This is also fueled by natural disasters such as a pandemic, due to which employees prefer working from home. With the rapid increase in usage of real-time applications, demands for meeting QoS agreements have increased proportionally [1]. IP networks provide best-effort service, i.e., all efforts are made to deliver data but with no guarantee of successful delivery. Real-time applications need a guarantee of timely arrival of data, which must be provided by the routers and not just the network edges (or the hosts) [2]. Therefore, it is necessary to implement QoS strategies at the network routers to ensure better user experience. In SDN, this is made possible by taking the QoS decisions at the controller, and enforcing the QoS policies at the switches using OpenFlow. To make efficient routing decisions, the delay should be monitored continuously.

In SDN, there are various approaches to measure delay [3]–[8]. Most of them are active measurement methods, i.e., they construct a specialized control packet (probe packet) and use OpenFlow protocol to inject the probe packet into the data plane. The probe packet physically travels the network links in the datapath and is forwarded back to the controller by a switch at the end of the path. The time taken by the packet to travel the path is used to calculate the delay. This approach suffer from: 1) Data plane footprint, due to injection of probe packets into the data plane, 2) monitoring overhead, due to the processing required at the controller to create probe packets and receive them, and 3) scalability issues, due to increase in the number of probe packets required with the number of switches, links and queues at the egress path. Details about the current approaches for delay measurement and their issues are discussed in Section II.

By studying network traces, authors in [9] have shown that queueing delay can be significant in todays networks. With this motivation, we propose a queueing delay monitoring (qMon) mechanism in OpenFlow based SDN. Many researchers as well as the Open vSwitch developers are working together to add features to Open vSwitch [5], [10] and enhance its capabilities. Therefore we take the liberty of slightly modifying the OpenFlow message to collect queue statistics. Queue statistics from Open vSwitch [11] are polled at regular intervals and queueing theory is applied at the controller to the aggregated queue statistics to obtain the estimated average waiting time of packets in the queue over a given time interval. Under the assumptions (discussed in Section IV), the estimated queueing delay can then be used to further estimate the link latency. This approach addresses most of the issues related to the active delay measurement techniques discussed earlier. The proposed approach relies on the queue statistics message, and no packets are injected into the data plane. Further, qMon can be integrated with existing traffic monitoring modules that poll for queue statistics to estimate traffic load. The proposed approach is scalable, as it requires sending only a single queue statistics request message for estimating link latency at each switch, irrespective of the number of transmit (TX) ports and queues at the TX ports at each switch. Thus, qMon is scalable

Manuscript received October 10, 2021; revised March 23, 2022; approved for publication by Sangheon Pack, Division 2 Editor, May 29, 2022.

S. Rathee, S. Tiwari, K Haribabu, and A. Bhatia are with Birla Institute of Technology and Science, CSIS Department, Pilani, Rajasthan. email: {p2015007, f2016935, khari, ashutosh.bhatia}@pilani.bits-pilani.ac.in.

S. Rathee is the corresponding author.

Digital Object Identifier: 10.23919/JCN.2022.000023

with respect to the size of the network. A detailed discussion on the issues related to the probe packet based methods, and how qMon addresses these issues are given in Section II.

Rest of the paper is organised as follows: Section II discusses the existing work on delay measurement and their issues. In Section IV, we formulate the problem of finding queueing delay as a single-queue single-server queueing problem and discuss how Little's law can be applied to estimate the queueing delay in an Open vSwitch datapath. It also discusses the batch means method of finding confidence intervals for the queueing delay. qMon is prototyped in Section V. In Section VI, we discuss the experimental setup used for evaluation of the proposed method. Section VII compares the delay trends between qMon and ping RTTs and evaluates the accuracy of the proposed method. Section VIII concludes the work.

II. Issues and Related Work

Delay measurements can be categorized into active and passive types [12]. Active delay measurement involves constructing a special timestamped control packet (called a probe packet) and injecting it into the datapath. At the end of the path, the receiving node estimates the path delay by calculating the transit time of the probe packet through the path. There are several issues associated with the active delay measurement methods in SDN:

1) *Data plane footprint:* The probe packets injected into the data plane consume the bandwidth of the links involved in the path. The bandwidth consumption increases with an increase in packet injection frequency, leading to a decrease in the available bandwidth and the possibility of change in the traffic behavior [13].
2) *Monitoring overhead and scalability:* Active delay measurement methods require creation of probe packets, injecting them into the data plane, and processing the probe packets at the switches on receiving them. Current active probing methods do not consider the possibility of multiple TC (traffic control) queues at the switch's egress ports. For measuring link delay for each of the TC queues, it would be required to send a separate probe packet through each of the queues. With the increase in the number of TC queues at the egress ports and the number of switches, the number of probe packets required increases multiple-folds, which in turn increases the processing overhead on the controller. When measuring end-to-end delay, the number of probe packets increases with an increase in the number of flows. In both the cases, probe packets might be queued at the controller on arrival [14], waiting to be processed. The waiting time of the probe packets at the controller leads to inaccurate link delay measurements.

Passive delay measurements revolve around 1) analytical/statistical methods to model the flow of traffic to estimate delay, or 2) time stamping the existing traffic in the datapath [12]. Limitations of passive methods are inaccuracies in estimating delay. This is due to the assumptions made while employing statistical methods to model the traffic, as it is challenging to have a single model that holds for all traffic distributions. Use of existing traffic in the data plane to measure delay is infeasible in OpenFlow enabled SDN, as it requires time stamping packets in the data plane and comparing their transit times across two end-points.

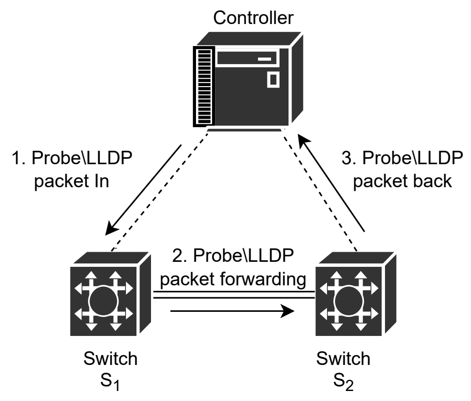


Fig. 1. Active delay monitoring in SDN networks.

Next, we discuss the existing works on measuring the delay in SDN using active and passive measurement methods.

A. Active Delay Monitoring Methods

Authors in [3]–[8] have proposed active delay monitoring schemes in SDN. The method proposed in OFMon [3] uses OpenFlow to send a time-stamped ethernet frame through a link (say, $S_1 - S_2$, where S_1 and S_2 are switches), and back to the controller, as shown in Fig. 1. The link delay is calculated by subtracting the summation of link delays between controller to switch S_1 and S_2, and a small processing offset introduced by the controller, from the total transit time of the probe packet. The link delay between controller and switches can be estimated by measuring the RTT of statistics request and statistics reply messages. Whereas, the processing offset is calculated for the underlying hardware by measuring the latency on an unused link. Although the size of the probe packet used (24 bytes) is less as compared to ICMP (196 bytes, request/reply 98 bytes each [3]), the number of packets injected into the data plane increases with the frequency of measurement, leading to an increase in data plane footprint. The method being an active probe method also suffers from active measurement issues such as monitoring overhead and scalability.

In [4], [5], authors use timestamped LLDP (link layer discovery protocol) packets (which are used for topology discovery by the SDN controllers) to measure the link delay. The method to calculate the link delay is mostly similar to the method described in OFMon [3]. LLDP-looping [5] is an extension of LLDP [4], which modifies Open vSwitch to enable looping of LLDP packets between source and destination to measure link RTT with high accuracy. While the methods reuse LLDP packets to reduce the data plane footprint, but since the LLDP packet generation is coupled to the controller's topology discovery module, thus their measurement frequency is limited. With an increase in the packet injection frequency, the method is similar to OFMon [3] and has the same limitations.

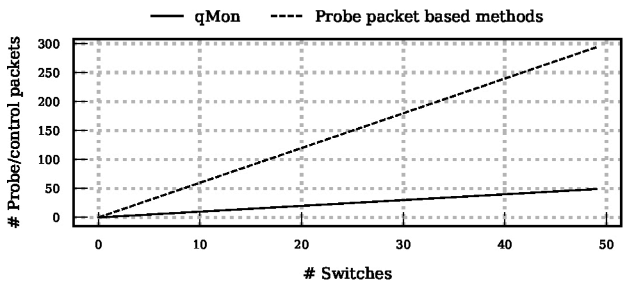


Fig. 2. Number of switches vs Number of probe/control packet.

OpenNetMon [6] is based on similar principles as OF-Mon [3], LLDP [4], and injects probe packets to measure the one-way end-to-end delay of a flow. GRAMI [7] inserts probe packets from selected nodes in the network to measure RTT for a single link or between any two switches in the network. It is resource efficient as it does not require involvement of the controller for online RTT monitoring and requires only four flow entries to be installed at every switch. TTL-Looping [8] actively measures end-to-end one-way delay with microsecond precision for flow by multiple iterations of the probe packet along the flow path. This is made possible by storing the required number of iterations in the probe packet's counter field. The flow entries along the path decrease the value of the counter and forward the packet to the next-hop switch. The last switch in the path reverses the direction of the packet. This continues until the counter value becomes zero and the packet matches the flow entry that sends it to the controller. The transit time of the packet is divided by the number of iterations to calculate the RTT. The RTT is then used to compute a one-way delay using the traffic proportion in each direction.

The method proposed in [15] uses OpenFlow to send a PACKET_IN packet (instead of a LLDP packet) through a link connecting the switches, and back to the controller, as shown in Fig. 1. The link delay is calculated by subtracting the summation of link delays between controller to source switch and destination switch w.r.t the link from the total transit time of the PACKET_IN packet. The link delay between controller and switches is estimated using by ECHO_REQUEST and ECHO_RESPONSE messages. LatencySmasher [16] estimates per link latency using time series model and compares the results with default controller delay measurements. It reduces the monitoring overhead but the results are not overlapping with the true delay. In [17], the authors proposed a mathematical solution to measure the transmission and propagation delay. They put a threshold on the bandwidth and assumed no queueing delay in the system. Whereas in a real network the queueing delay varies depending on the network traffic.

B. Passive Delay Monitoring Methods

To the best of our knowledge, not much work has been done on passive delay measurements in SDN. Authors in [18] propose a queueing model for measuring average queueing delay of TCP flows in SDN. The parameters required as input to the method, such as - packet length, buffer size, and link bandwidth, can be computed by querying the switches through OpenFlow. With a growing demand for streaming services, which use UDP as the underlying protocol, the network traffic might contain a high percentage of UDP flows or a heterogeneous mix of multiple flows. Method proposed in [18], however, works only for multiple TCP flows. Approaches like [19] assume that the inter-arrival time of the packets is exponentially distributed, and propose a queueing model for estimating end-to-end delay. However, this method focuses on modeling network nodes for computing the required parameters for network design and planning, and does not propose a method for real-time delay monitoring. In SDN networks default configuration for communication is without QoS support, thus it can affect the end-to-end performance of network services and applications [20], [21].

In this paper, we propose a passive method to measure link delay in SDN networks, without making any assumptions about the traffic distribution. The proposed method solves the issues stated earlier. It depends entirely on the queue statistics messages from the switches and, as a result, has zero data-plane footprints. Monitoring overhead is reduced considerably as estimation of queueing delay only requires an application of the Little's formula over the queue statistics received. Delay calculation can, therefore, be implemented inline with other QoS related modules, and does not require a dedicated controller module. The controller is now exempt from creating probe packets and receiving them, thus considerably reducing the processing overhead. Queue statistics message received from a switch includes the queueing information for all the queues at each of the ports. Therefore, queueing delay for each of the queues can be computed with the information from a single queue statistics message. Thus, the proposed method is scalable, as it requires sending of a single queue-statistics request message to each of the switches to monitor queueing delay at all the links in the network. This is in contrast to the existing active probe packet based methods which require sending a probe packet to each of the output port queues in all the switches to measure the queueing delay. Fig. 2 shows the number of probe packets injected by active probe based methods versus the number of control packets required in qMon to measure the queueing delay at all the output ports for a given number of switches in the network.

III. Background

We apply queueing theory to model the flow of packets through Open vSwitch [11] on a linux datapath[1].

The linux TC consists of various queueing disciplines (qdiscs) which fall under two categories: Classless and classful. The default queueing discipline in the traffic control is pfifo_fast [22], which can be changed through userspace TC utility. Classful qdiscs are flexible, that is, classful or classless child qdiscs can be attached to them, and can share bandwidth with other classful qdiscs, when possible. Leaf

[1]To evaluate proposed method, we have used Linux OVS tree datapath, provided by Open vSwitch v2.11.1.

classes have a classless qdisc attached to them. The queues managed by the classless qdiscs (attached to classful qdiscs) are where the packets finally get enqueued or dequeued, by the algorithm corresponding to that class. Examples of classful qdiscs are HTB (hierarchical token bucket) and CBQ (class based queueing) [23]. Classless qdiscs are elementary qdiscs and are rigid in the sense that they cannot have children, nor can they share bandwidth with other classes. They maintain a queue, from which the packets get enqueued or dequeued by the algorithm corresponding to the qdisc. Examples of classless queueing disciplines are: pfifo, bfifo, TBF, SFQ, pfifo_fast (default used by linux TC) [23].

Linux-htb (from now on referred to as HTB) [24] is a classful queueing discipline, which means child qdiscs can be attached to it, which forms a tree like structure. Each HTB class has a token bucket associated with it, which is filled with tokens at a rate which is determined by the rate assigned to the class. To dequeue a packet, HTB charges certain amount of tokens proportional to the size of the packet from the bucket associated with the class. If there are not enough tokens, then the class tries to borrow tokens from the sibling classes. If enough tokens are not available, the packet has to wait until the bucket has enough tokens. Thus, bandwidth is limited by throttling the packets, which is also known as shaping.

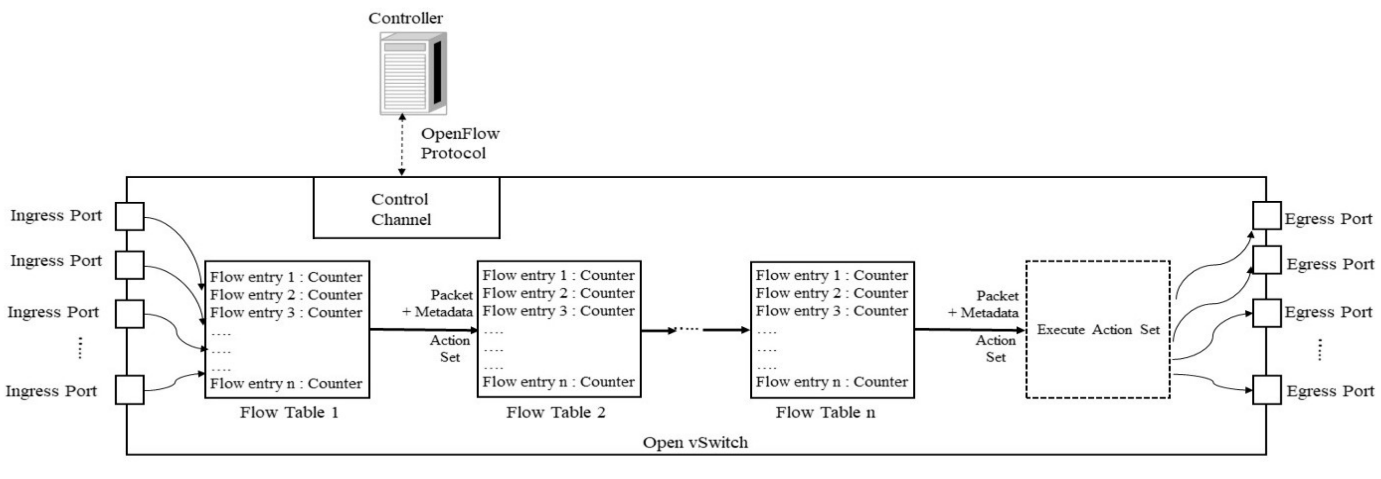
When a packet originating from a child class is dequeued from its parent class, a number of tokens depending on the size of the packet is charged from the bucket of the parent class. These tokens then will not be available for packets originating from other child classes. This ensures that child classes can not have a rate more than their parent class. While enqueueing and dequeueing, the kernel interacts only with the root qdisc. To enqueue a packet, kernel calls `htb→enqueue()` located in `/net/sched/sch_htb.c`, which further calls `htb→htb_classify()` to classify the packet into one of the child HTB classes. It walks the tree and enqueues to a classless qdisc attached to one of the leaf nodes. To dequeue a packet, the kernel calls `dequeue()` on the root qdisc, which calls dequeue function of the child classes, and so on, until a packet is dequeued from the leaf class, and is passed on to the parent classes, charging tokens from the buckets in the process, until it is dequeued from the root qdisc. The dequeued packet is then placed onto the driver queue for transmission. Open vSwitch does not support multiple levels of hierarchy in classful queueing disciplines. It is therefore only possible to configure multiple HTB queues at the first level. Borrowing of tokens can still take place among the queues at the first level. By default, the leaf nodes have a pfifo qdisc attached to them, with a queue length of `txqeueuelen`, which is a parameter associated with each interface. It has a default value of 1000 packets, which can be changed with the `iproute2` userspace utility. The queue length plays a significant role during burst traffic, i.e., when there is a large amount of traffic in a short period of time, this can lead to queues getting filled up in a very short span of time. If a queue gets filled up beyond its maximum size then the incoming packets are dropped.

IV. SYSTEM MODEL

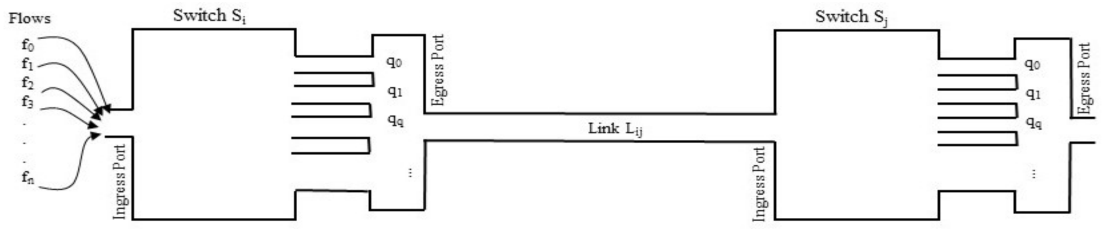
We consider an SDN network with Open vSwitches. Fig. 3(a) provides the internal details of the switch and controller configuration. Every switch has multiple ports each connected to host, switch or controller and every egress port supports multiple queues (maximum number of queues per port depends on the vendors) as shown in Fig. 3(b). We consider an out-of-band controller configuration, the controller is directly connected to every switch in the network. Open vSwitch supports multiple forwarding tables and each forwarding table consists of multiple flow entries. In SDN, the first packet of a flow may not match any of the flow entries and as a result the packet is sent to the controller. Now the controller estimates the type of flow and establishes a path for the flow through the network by installing flow entries on all the switches in the path. It also specifies the flow to output queue mapping in the action field of the flow entry. The proposed solution needs only per queue statistics to measure the delay. Controller can poll the per queue statistics using *QueueStatsRequest* and can measure the queueing delay for each queue. The solution is not dependent on the number of the queues configured on the egress port.

We define link delay as the time required to transmit a packet in its egress path, which is the sum of processing, queueing, transmission and propagation delay. Throughout this paper, we assume, that processing, transmission, and propagation delays remain constant. Most of the modern routers are capable of processing the packets at almost the line rate. In SDN networks, a packet may not match any flow entry and is consequently sent to the controller (slow path). Although such packets experience larger processing delays than usual, they are usually the first packet of a flow and help the controller in establishing flow entries on the switches in the flow path. Therefore delay experienced by such packets is not representative of the delays experienced by majority of the packets that match the flow entries and are forwarded through the fast path. It is therefore assumed that each packet takes almost the same amount of time to get processed. Transmission delay is the time required to inject all the bits of a packet so that it can be transmitted over the physical medium, and is a function of the packet length. Assuming that the packet length remains constant on an average, transmission delay can also be assumed to be constant. Propagation delay is a function of the path length, which remains constant for a given system configuration.

Queueing delay is the time for which a packet is enqueued in a queue before it is transmitted. Fig. 4 illustrates the ingress and egress path of a packet through Open vSwitch on a linux datapath: Packets are queued at the network interface card (NIC) ring buffer on receive (RX) path, ingress queue of the RX interface, egress queue of the transmit (TX) interface and the NIC ring buffer on the TX path. The NIC ring buffer is also known as the driver queue or the direct memory access (DMA) ring buffer and is a part of the NIC. The ingress and egress queues are a part of the TC subsystem of the linux kernel [25], [26]. TC allows traffic shaping at the egress queues, while only traffic policing is possible at the ingress



(a)



(b)

Fig. 3. (a) Open vSwitch (b) detailed diagram of two switches directly connected to each other.

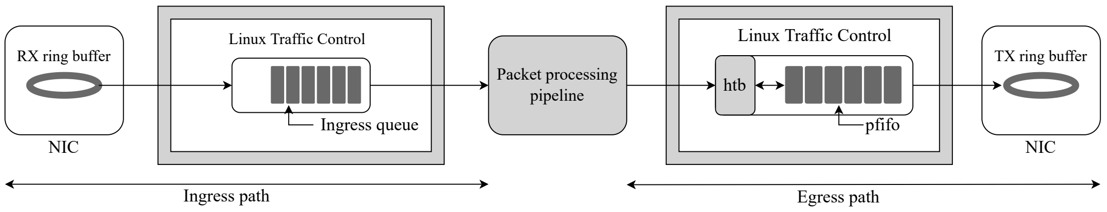


Fig. 4. Queues in the ingress and egress path of the packet.

queues. It is assumed that for most of the time, the CPU is capable of processing the packets at the input port so that there is no buffering at the ingress queue. It is also assumed that the TC egress queues are configured so that the total bandwidth allocated to them does not exceed the line-rate of the underlying physical layer. Under this assumption, packets are buffered at the TC queues and no buffering takes place at the egress driver queue. Link latency due to buffering at the TC queues is a function of the traffic distribution arriving at the queues. The next section summarises the queueing of packets in linux traffic control and the linux-htb queueing discipline to illustrate the same.

A. Queueing Model

Queueing delay is a function of the packet arrival and service rate distributions. Due to the nature of the packet scheduler (HTB), the service rate distribution is a function of the packet size. Internet traffic, however, is dynamic in nature which makes it tough to categorize the traffic into a known distribution [27]. Therefore, our objective is to model the TC queues as a single queue and a single server queueing problem, and solve it for a general packet arrival and service rate distribution using Little's law,

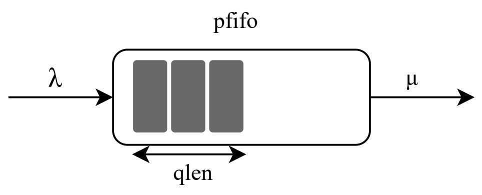


Fig. 5. A TC queue.

$$L = \lambda W, \tag{1}$$

where L is the average number of packets in the queue, λ is the average arrival rate, and W is the average waiting time.

Fig. 5 shows a TC queue modelled as a single queue, single server queueing system. To meet end-to-end QoS requirements of latency-sensitive applications, it becomes necessary to estimate delay at short intervals of time. So that the routing decisions can be taken to adapt the current load conditions and QoS requirements. Let the polling interval for the queue-statistics be t. Let t_1 and t_2 be two consecutive polling epochs.

Our objective is to obtain the average waiting time of packets in the finite time interval $[t_1,t_2]$, where $t_2 - t_1 = t$.

Given λ_{av} as the average rate of arrival of packets in the queueing system, average queue length (Len) as the average number of packets in the system over infinitely large time, and assuming that Len_{av} is less than the queue size (i.e., no packet loss due to filling of queue), the average waiting time for packets in the queue can be obtained by using Little's law as $\overline{W} = Len_{av}/\lambda_{av}$. The equality however does not necessarily hold true over finite time intervals due to interval edge effects [28]. Let $\overline{Len}(t_1,t_2)$ be the mean queue length, $\overline{\lambda}(t_1,t_2)$ be the mean arrival rate of the packets, and $\overline{W}(t_1,t_2)$ be the mean waiting time of a packet in the queue in the interval $[t_1,t_2]$. Authors in [28] regard $\overline{Len}(t_1,t_2)/\overline{\lambda}(t_1,t_2) = \overline{W}(t_1,t_2)$ as an estimator of the underlying true average. They also suggest taking a statistical approach to estimate the waiting times, and using the method of batch means to apply the estimator $\overline{W}(t_1,t_2)$ to estimate a confidence interval for $\overline{W}(t_1,t_2)$. Section IV-C discusses how to apply batch means method.

B. Estimator for $\overline{W}(t_1,t_2)$

Our estimation of waiting time relies on queue statistics obtained by sending an OpenFlow queue-stats request message from controller to the switches. `OpenFlow v1.3` queue-stats reply message has a counter of the number of transmitted packets (*tx_packets*) through the queue. Let t_1 and t_2 be two consecutive times at which the controller receives the queue statistics. For a stable system, $\overline{\lambda}(t_1,t_2)$ is equal to the mean throughput over the interval $[t_1,t_2]$, and can be estimated using (2).

$$\overline{\lambda}(t_1,t_2) = (tx\_packets_{t_2} - tx\_packets_{t_1})/(t_2 - t_1). \quad (2)$$

Queue statistics reply message however does not contain any field with the queue length information. We observed that the current length of each queue is maintained in the TC subsystem of the kernel. By modifying `netdev-linux.c` in Open vSwitch 2.11, queue statistics reply message can be modified to include queue length. As such, the queue statistics message reply after modification contains the following fields: $\langle queue\_id, tx\_bytes, tx\_packets, qlen, errors, time\rangle$. $\overline{Len}(t_1,t_2)$ can then be obtained in the following way:

$$\overline{Len}(t_1,t_2) = (Len_{t_1} + Len_{t_2})/2. \quad (3)$$

Estimator for $\overline{W}$, in the interval $[t_1,t_2]$ can then be obtained by Little's law as:

$$\overline{W}(t_1,t_2) = \overline{Len}(t_1,t_2)/\overline{\lambda}(t_1,t_2). \quad (4)$$

C. Confidence Intervals for $\overline{W}(t_1,t_2)$

Consider m consecutive estimation intervals $[t_0,t_1],[t_1,t_2],\cdots,[t_{m-1},t_m]$. Let the estimated waiting times in the intervals be $\overline{W}(t_0,t_1), \overline{W}(t_1,t_2),\cdots,\overline{W}(t_{m-1},t_m)$. Confidence intervals for the mean waiting time over the union of the intervals $[t_0,t_1] \cup [t_1,t_2] \cup \cdots \cup [t_{m-1},t_m] = [t_0,t_m]$, $\overline{W}(t_0,t_m)$ can be calculated by applying the formulas given by [28]. We consider batches of size m = 5, by taking m consecutive intervals. The 90% confidence interval can then be calculated by formulas (24) in [28]. The corresponding formulas for the current use case is given by (5) below. It is based on Student's t-distribution. For $t_{0.025,m-1} = t_{0.025,4} = 2.132$.

$$\begin{gathered}\overline{W}(t_0,t_m) = \frac{1}{m}\sum_{k=1}^{m}\overline{W}(t_{k-1},t_k)\\ \overline{S}_{(m)}(t_0,t_m) = \frac{1}{m-1}\sum_{k=1}^{m}(\overline{W}(t_{k-1},t_k) - \overline{W}(t_0,t_m))^2, \quad (5)\\ \Big[\overline{W}(t_0,t_m) \quad - \quad \frac{t_{0.025,m-1}S_m(t_0,t_m)}{\sqrt{m}},\\ \overline{W}(t_0,t_m) + \frac{t_{0.025,m-1}S_m(t_0,t_m)}{\sqrt{m}}\Big].\end{gathered}$$

D. Calculation of Calibration Constant and Delay

Assuming that packet's length and the system configuration remain constant; processing delay, transmission delay and propagation delay remain constant and can be calculated as constants by sending probe packets through the empty link. Let $\overline{D}(t_1,t_2)$ be the estimator for the mean link delay $\overline{D}(t_1,t_2)$ in the interval $[t_1,t_2]$. $\overline{D}(t_1,t_2)$ is given by (6).

$$\overline{D}(t_1,t_2) = \overline{W}(t_1,t_2) + C, \quad (6)$$

where C is the calibration constant. The confidence interval for $\overline{W}(t_1,t_2)$ and hence, $\overline{D}_{(t_1,t_2)}$ can be found out by batch interval method as discussed.

For demonstration purpose, consider Fig. 1. To measure calibration constant, C, for the link $S_1 - S_2$ a probe packet is sent from controller (C_0) to switch S_1. Flow entry at switch S_1 sends the probe packet on the link $S_1 - S_2$. From switch S_2 the packet is sent back to the controller. The total time taken by the probe packet to traverse $C_0 - S_1 - S_2 - C_0$ is calculated by subtracting the arrival and the departure time of the packet at the controller. To obtain the calibration constant for the link $S_1 - S_2$ the control link delays, $C_0 - S_1$ and $C_0 - S_2$, are subtracted from the total transit time of the probe packet. Control link delay between the controller and a switch is calculated by sending a statistics request message from the controller to the switch. The time elapsed between sending of the stat request message and receiving the corresponding reply at the controller is halved (assuming symmetric delays) to get the one way delay on control link. The control link delays, $C_0 - S_1$ and $S_2 - C_0$, are subtracted from the total time to traverse $C_0 - S_1 - S_2 - C_0$ to get the delay on link $S_1 - S_2$. The calibration constant is obtained by averaging over the delays obtained from several iterations of the method discussed above. Similar procedure is used to obtain the calibration constant for the experimental topology.

V. PROTOTYPING

The proposed queueing delay monitoring approach, qMon, requires a slight modification of Open vSwitch v2.11.1 and Ryu running on Ubuntu 18.04 LTS as the underlying operating system. Open vSwitch leverages `RTNETLINK` to communicate with the TC subsystem

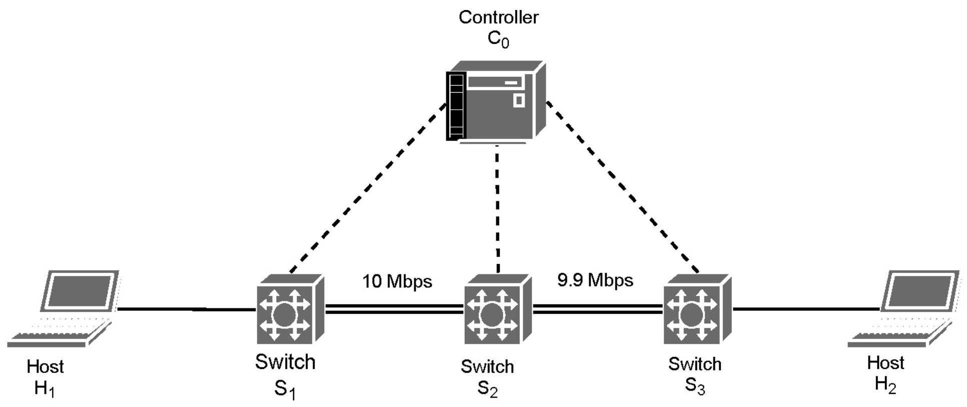


Fig. 6. A topology with 3 switch, 2 hosts and controller connected in a linear fashion for evaluating the proposed method.

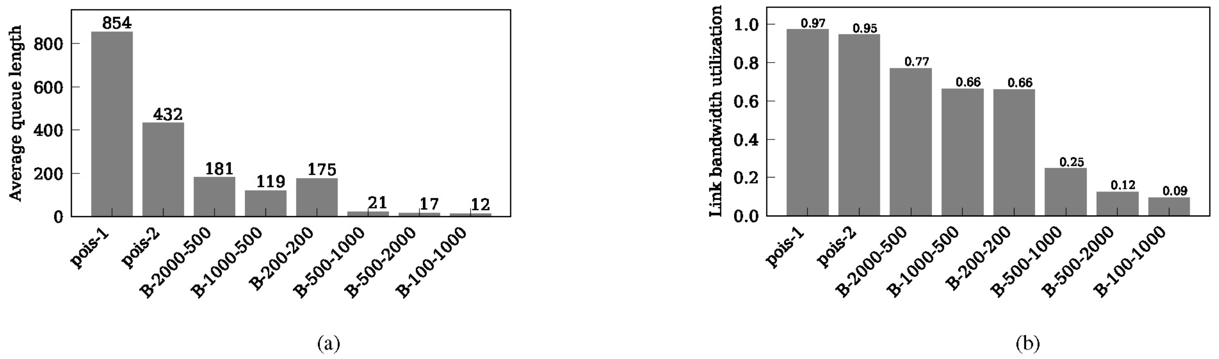


Fig. 7. (a) Average queue length (b) average link bandwidth utilization (ρ) for the traffic scenarios considered.

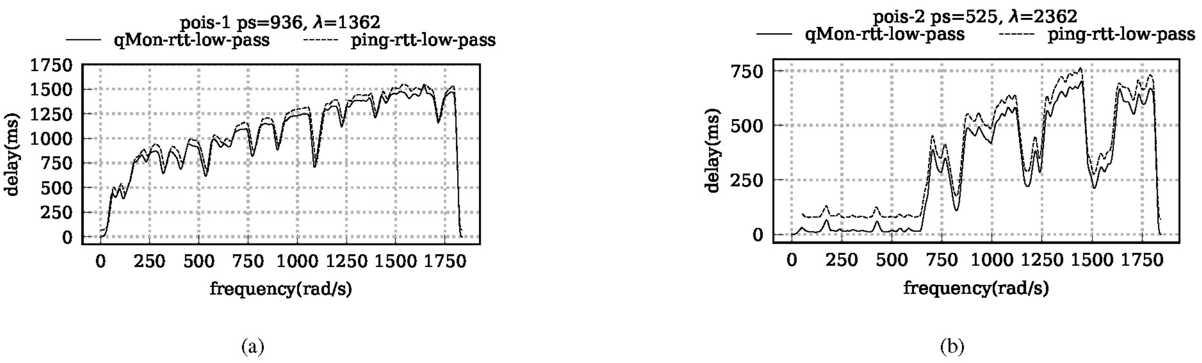


Fig. 8. Low pass delay signals for qMon versus ping RTT for Poisson traffic, ps = packet size, λ = mean packet arrival rate. Ping RTT signal is offset by +70 ms for better visibility.

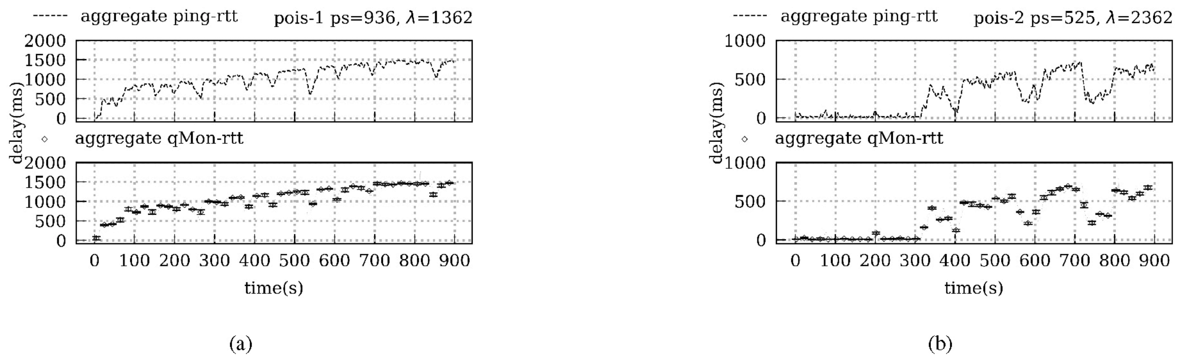


Fig. 9. 90% confidence interval for qMon RTT for Poisson traffic, ps = packet size, λ = mean packet arrival rate.

TABLE I
PEARSON CO-EFFICIENT FOR ALL THE EXPERIMENTS.

Experiment name	Correlation coefficient
pois-1	.99
pois-2	.99
B-2000-500	.97
B-1000-500	.87
B-200-200	.84
B-500-1000	.60
B-500-2000	.74
B-100-1000	.32

of the Linux kernel to add/remove or get a queueing discipline [29]. To get queue-statistics, ovs switch makes a `RTNETLINK` request message (`RTM_GETQDISC`). The corresponding `RTNETLINK` reply received from the kernel is parsed into `struct gnet_stats_basic` and `struct gnet_stats_queue` defined in `linux/gen_stats.h`. The struct netdev_queue_stats defined in `ovs/lib/netdev.h` represents queue-stats reply. OpenFlow 1.3 queue-stats reply does not have a field for queue length. Open vSwitch v2.11.1 therefore does not include a field for queue length in struct netdev_queue_stats. We observed that for experimental purposes, we can add a queue length field (`‘uint64_t qlen’`) to `struct netdev_queue_stats`. The field `qlen` is populated with queue length received in the `struct gnet_stats_queue` and is propagated up to the controller as the queue-statistics reply.

The implementation of qMon does not require any modification in the packet processing pipeline of Open vSwitch. Enterprise and academicians often like to customize SDN to suit their specialised requirements for QoS and security. If the target switches are hardware switches, it only requires changes to the software/firmware of the switches to support qMon. Enterprise infrastructure administrators/engineers can specify their specific customization to to the manufacturers and request them to customize the software/firmware of the switches to meet their needs for QoS optimizations.

Since the queue-stats reply is modified to include queue length, minor modifications are made to Ryu controller, so that it can recognize the new field as queue length. The delay monitoring module in the controller makes queue-statistics request messages to all the switches in the network at fixed intervals. We fixed the time interval to 500 ms. For the time interval $[t_1, t_2]$, (where $t_2 - t_1 = 500$ ms) the module uses the queue statistics received at time t_1 and t_2 to estimate the mean queueing delay for the interval $[t_1, t_2]$. The module can optionally store the estimated mean queueing delay for the last m intervals to calculate a 90% confidence interval for the union of these intervals. In [28] the authors suggest the value of m be equal to 5, which is the value considered in the evaluation section.

VI. EXPERIMENTAL SETUP AND EVALUATION CRITERIA

We evaluate the performance of qMon on a testbed consisting of three systems installed with the patched version of Open vSwitch connected in a linear topology. A host is connected to each edge switch, and all the switches are directly connected to the controller. The topology for the testbed is shown in Fig. 6. The link bandwidth is set to 10 Mbps on link S_1–S_2 and 9.9 Mbps on link S_2–S_3. Bandwidth for link S_2–S_3 is less than link S_1–S_2 to ensure buffering of packets on link S_2–S_3. Each of the systems has the following capabilities - Processor: Intel(R) Core(TM) i5-4590 CPU @3.30GHz, RAM: 32 GiB (8 GiBx4), OS: Ubuntu 18.04 LTS (Linux 4.15.0-91-generic), NIC: 1 Gbits/sec between two switches or hosts, 100 Mbits/sec between controller and switches.

We use D-ITG [30], to generate traffic from host h1 to host h2. For each experiment, the traffic rate is set at a value to enable moderate buffering at TC queues. The evaluation is performed over two kinds of traffic distributions: bursty with exponential ON/OFF periods and Poisson with exponentially distributed packet length. For bursty traffic, packet length is set to 512 bytes with varying ON/OFF ratio in different experiments. The experiments are named as $B-X-Y$, where B means bursty traffic, X is ON time, and Y is OFF time in ms. For Poisson traffic, mean packet length is set to 930 bytes and 512 bytes for experiments `pois-1` and `pois-2` respectively. However, since RTT measurements are running parallelly, the observed mean packet size and arrival rates are 936 and 1362, 525, and 2362 for `pois-1`, and `pois-2` experiments respectively.

The delay per link is estimated using qMon. The round trip time is estimated by adding the estimated link latencies at the links S_1–S_2, S_2–S_3, S_3–S_2, S_2–S_1. The delay trends obtained from the qMon RTT are compared with ping RTT measurements. Ping has been used as a standard tool for RTT measurements in IP networks. It is therefore a reliable measure of RTT. Although ping measurements are active RTT measurements, it will not change the traffic behavior in the current setup, since a traffic generator is being used to generate traffic of a fixed distribution.

It is complicated to compare two-time series data due to the high frequency of variations in measurements. It is, therefore, desirable to visualize and compare the low-frequency component of the delay variations. We consider the delay measurements as discrete signals and apply a low pass filter to the delay signals. This will help us to compare the low-frequency components of qMon and ping RTT signals. We use `windowed-sinc` as our low pass filter [31]. For constructing the filter, two parameters are required: Cut-off frequency(f_c) as a fraction of the sampling rate and transition band (b) as a fraction of the sampling rate. The length of the filter N can be determined by the formula given in (7).

$$N = 4/b. \tag{7}$$

The low frequency signals are obtained by convulsing the sinc function(c_0(n)) with the delay signal. The sinc function is constructed as shown in (8), (9), and (10).

$$c_1(n) = sinc(2 * f_c * (n - (N-1)/2)), \tag{8}$$

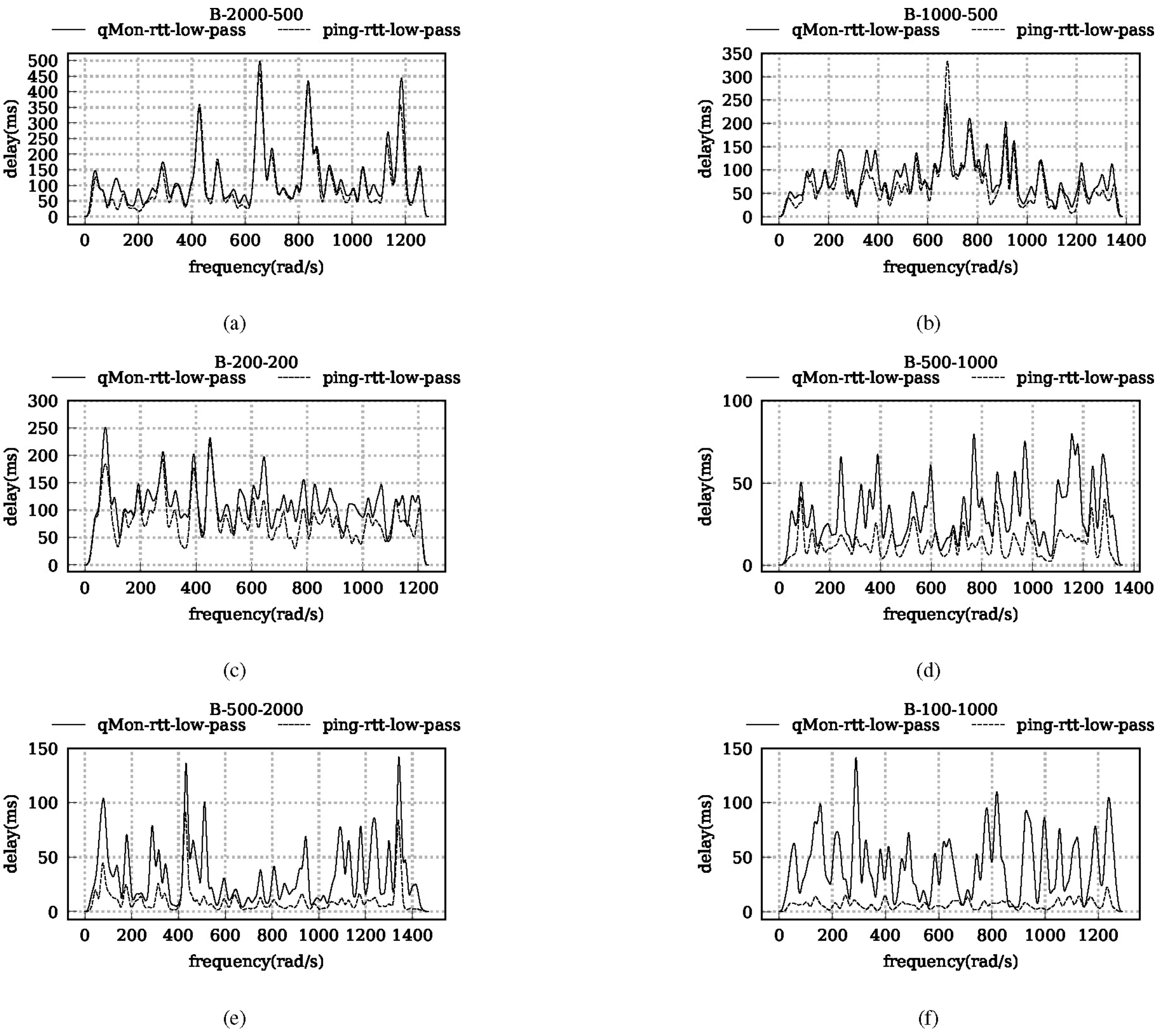


Fig. 10. Low pass delay signals for qMon versus ping RTT for bursty traffic.

$$window(n) = 0.42 - 0.5 * cos(2 * pi * n/(N-1)) + 0.08 * cos(4 * pi * n/(N-1)), \quad (9)$$

$$c_0(n) = c_1(n) * window(n). \quad (10)$$

We set the cut-off frequency (f_c) at 1% of the sampling rate and the transition bandwidth (b) at 8% of the sampling rate.

Quantitative evaluation of the trend match between qMon and ping RTTs has been done by comparing Pearson's correlation coefficient between the low-pass delay signals. Pearson's correlation coefficient measures the linear synchrony between two signals. Values between -1 and 0 indicate a negative correlation, 0 and $+1$ indicate a positive correlation. In the current use case, it is already known that there is no implicit co-relation between qMon and ping RTT signals. However, the correlation values are an indication of the similarity between the signals. Figs. 7(a) and 7(b) compare the average queue length and average link bandwidth utilization for the experiments conducted.

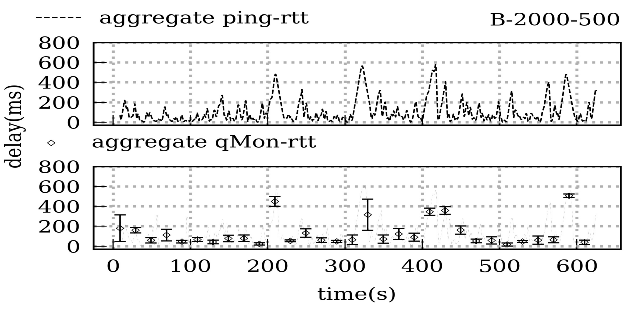
VII. RESULTS

A. *Poisson Traffic*

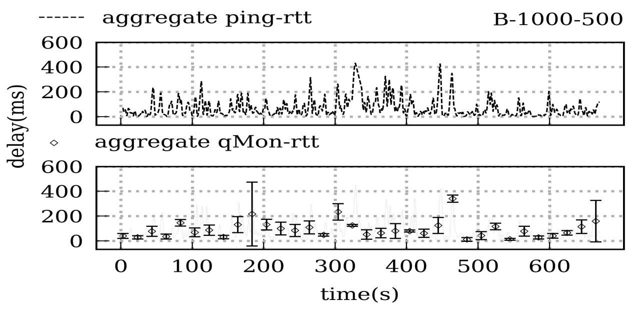
It can be observed from Figs. 8(a) and 8(b) that low frequency qMon and ping RTT signals in traffics `pois-1` and `pois-2` closely coincide. The ping RTT signals in Fig. 8 have been offset by +70 ms for better visibility of qMon and ping RTT signals. Table I shows that the correlation values for `pois-1` and `pois-2` traffics are quite high ($\geq .99$). The trend match can be attributed to the fact that queueing system is in steady state. Since the input traffic rate is Poisson distributed and packet sizes are exponentially distributed, the queueing system can be approximated as M/M/1/K with K (buffer capacity) = 1000 packets. The queueing system therefore reaches a steady state with Poisson distributed traffic. Smaller value of 90% confidence intervals in Figs. 9(a) and 9(b) also indicate that average waiting time estimations are fairly accurate in case of Poisson traffic.

B. *Bursty Traffic*

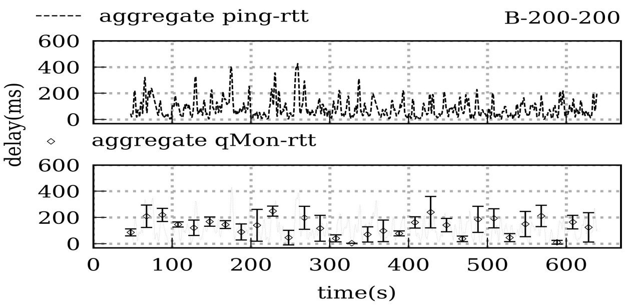
It can be observed from Figs. 10(a) and 10(b) that low frequency qMon and ping RTT signals in `B-2000-500`



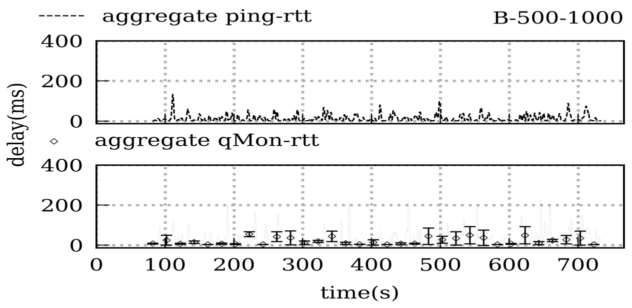
(a)



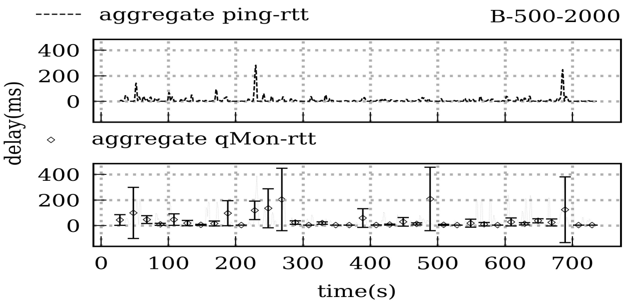
(b)



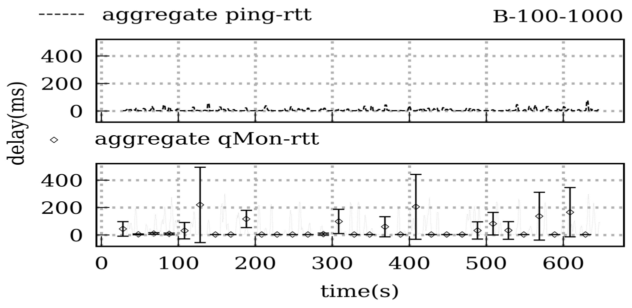
(c)



(d)



(e)



(f)

Fig. 11. 90% confidence interval for qMon RTT for bursty traffic.

and `B-1000-500` closely coincide. Table I shows that the correlation values for `B-2000-500` and `B-1000-500` are quite high. For `B-200-200`, `B-500-1000`, `B-500-2000` and `B-100-1000`, the correlation values decrease with a decrease in the ratio of ON/OFF periods, with the exception of `B-500-2000`. High correlation values for `B-2000-500` and `B-1000-500` can be attributed to the steady state of the queueing system due to generation of traffic at a constant rate during ON periods. Since the ON periods are of larger duration than OFF periods for `B-2000-500` and `B-1000-500`, the queue is in a steady state for a larger duration leading to accurate waiting time estimations. As the ratio of ON/OFF period decreases, the duration for which the queue is in a steady state decreases (Table I), with the exception of `B-500-2000`. Higher correlation value of `B-500-2000` than `B-500-1000` may be a result of (1) same length of ON periods and measurement interval (500 ms), which might lead to synchronization of OFF periods and the measurement epochs. Probability of this happening is less because ON/OFF periods are exponentially distributed, and (2) Longer mean OFF period (2000 ms) in `B-500-2000` than in `B-500-1000` (1000 ms). Longer OFF periods lead to steadiness in the system, leading to higher correlation between estimated and measured RTT. Low correlation value in `B-100-1000` is expected as the ON period (100 ms) is very less as compared to the measurement interval (500 ms), because (1) even though the ON/OFF periods are exponentially distributed, because of very low mean ON period (100 ms) there is a high probability of measurement epochs synchronizing with the OFF periods. (2) Even when the measurement intervals are synchronized with the ON periods, the burst of traffic happens only for 100 ms, but the average of all burst packets is taken over a time interval of 500 ms, leading to a inaccurate and numerically low estimated TX_rate. The numerically low estimated TX_rate leads to very high estimated values of waiting time by Little's law. This is apparent on observing Fig. 10(f). Figs. 11(a)–11(f) show the 90% confidence interval plots for experiments with bursty input. Confidence interval plots indicate a similar trend. `B-2000-500` and `B-1000-500` have narrow confidence intervals on an average as compared to `B-200-200`, `B-500-1000`, `B-500-2000` and `B-100-1000`.

VIII. Conclusion

We proposed qMon, a scalable and low overhead queueing delay monitoring method. However, the method does not work well for the bursty traffic with burst interval smaller than the current polling interval. To obviate this issue, the approach can be extended to maintain a $< queuelength, timestamp >$ vector in the switch itself which is updated at very short intervals of time. The controller can then poll for queue length vector at larger intervals and still construct a more accurate view of the queueing delay experienced by the packets. The idea although promising, requires a rigorous experimental evaluation, and we defer it for future work. Although we have demonstrated qMon on an OpenFlow based network, the technique can be applied in any network architecture which is capable of polling queue length information from switches/routers to a central location capable of processing the information.

References

[1] M. Karakus and A. Durresi, "Quality of service (qos) in software defined networking (sdn): A survey," *J. Netw. Comput. Applicats*, vol. 80, pp. 200–218, 2017.
[2] L. L. Peterson and B. S. Davie, *Computer networks: A systems approach*. Elsevier, 2007.
[3] K. Phemius and M. Bouet, "Monitoring latency with OpenFlow," in *Proc. IEEE CNSM*, 2013.
[4] L. Liao and V. C. Leung, "LLDP based link latency monitoring in software defined networks," in *Proc. IEEE CNSM*, 2016.
[5] L. Liao, V. C. M. Leung, and M. Chen, "An efficient and accurate link latency monitoring method for low-latency software-defined networks," *IEEE Trans. Instrum. Meas.*, vol. 68, no. 2, pp. 377–391, 2019.
[6] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow software-defined networks," in *Proc. IEEE NOMS*, 2014.
[7] A. Atary and A. Bremler-Barr, "Efficient round-trip time monitoring in OpenFlow networks," in *Proc. IEEE INFOCOM*, 2016.
[8] V. Altukhov and E. Chemeritskiy, "On real-time delay monitoring in software-defined networks," in *Proc. IEEE MoNeTeC*, 2014.
[9] A. Csoma, L. Toka, and A. Gulyás, "On Lower Estimating Internet Queuing Delay," in *Proc. IEEE TSP*, 2015.
[10] B. Pfaff *et al.*, "The design and implementation of open vswitch," in *Proc. USENIX NSDI*, 2015.
[11] B. Pfaff *et al.*, "The Design and Implementation of Open vSwitch," in *Proc. USENIX NSDI*, 2015.
[12] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software defined network traffic measurement: Current trends and challenges," *IEEE Instrum. Meas. Mag.*, vol. 18, no. 2, pp. 42–50, 2015.
[13] S. Zander, G. Armitage, T. Nguyen, M. Lutz, and B. Tyo, "Minimally intrusive round trip time measurements using synthetic packet-pairs," Tech. Rep. 060707 A, Centre for Advanced Internet Architectures, Swinburne University of Technology, 2006.
[14] "Ryu SDN framework using OpenFlow 1.3," [Online] Available: https://osrg.github.io/ryu-book/en/Ryubook.pdf.
[15] P. Sossalla, J. Rischke, and F. H. Fitzek, "Enhanced one-way delay monitoring with openflow," in *Proc. IEEE ICOIN*, 2022.
[16] M. Rahouti, K. Xiong, Y. Xin, and N. Ghani, "Latencysmasher: A software-defined networking-based framework for end-to-end latency optimization," in *Proc. IEEE LCN*, 2019.
[17] K. Venkatesh, L. Srinivas, M. M. Krishnan, and A. Shanthini, "QoS improvisation of delay sensitive communication using SDN based multipath routing for medical applications," *Future Generation Comput. Syst.*, vol. 93, pp. 256–265, 2019.
[18] M. Haiyan, Y. Jinyao, P. Georgopoulos, and B. Plattner, "Towards SDN based queuing delay estimation," *China Commun.*, vol. 13, no. 3, pp. 27–36, 2016.
[19] W. Li, J. Yang, and D. Zhang, "A method to calculate queuing delay for real-time services in IP networks," pp. 1–4, 2010.
[20] T. Chin, M. Rahouti, and K. Xiong, "Applying software-defined networking to minimize the end-to-end delay of network services," *ACM SIGAPP Appl. Comput. Review*, vol. 18, no. 1, pp. 30–40, 2018.
[21] M. Rahouti, K. Xiong, T. Chin, and P. Hu, "SDN-ers: A timely software defined networking framework for emergency response systems," in *Proc. IEEE SCOPE-GCTC*, 2018.
[22] [Online] Available: https://lartc.org/manpages/tc-pfifo_fast.pdf.
[23] "Linux advanced routing & traffic control HOWTO," [Online] Available: https://www.lartc.org/lartc.html.
[24] [Online] Available: https://lartc.org/manpages/tc-htb.pdf.
[25] L. Angrisani, G. Ventre, L. Peluso, and A. Tedesco, "Measurement of processing and queuing delays introduced by an open-source router in a single-hop network," *IEEE Trans. Instrum. Meas.*, vol. 55, no. 4, pp. 1065–1076, 2006.
[26] W. Almesberger *et al.*, "Linux network traffic control—implementation overview," 1999, [Online] Available: http://marco.uminho.pt/disciplinas/ST/traffic_control_on_linux.pdf.
[27] S. Floyd and V. Paxson, "Difficulties in simulating the Internet," *IEEE/ACM Trans. Netw.*, vol. 9, no. 4, pp. 392–403, 2001.
[28] S.-H. Kim and W. Whitt, "Statistical analysis with Little's law," *Operations Research*, vol. 61, no. 4, pp. 1030–1045, 2013.
[29] "Linux Programmer's Manual," [Online] Available: http://man7.org/linux/man-pages/man7/rtnetlink.7.html.
[30] A. Botta, A. Dainotti, and A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios," *Comput. Netw.*, vol. 56, no. 15, pp. 3531–3547, 2012.
[31] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*. USA: California Technical Publishing, 1997.

Sandhya Rathee completed her doctorate from Birla Institute of Technology and Science, Pilani in 2021 under the supervision of Dr. K Haribabu and Dr. Ashutosh Bhatia. She got CSIR - International travel grant for a conference in Japan. Also, she had been selected for Indo German Challenge for Sustainable Production - an innovative program between BITS, Pilani, India and Technical University (TU), Braunschweig, Germany. She is an active reviewer of Computer Networks, Elsevier. Her research interests lie at the intersection of computer networks and software defined networking.

Shubham Tiwari graduated with a B.E. Computer Science and M.Sc. Mathematics from Birla Institute of Technology and Science, Pilani in 2021. His interests are systems and networking.

Haribabu K (Senior Member, IEEE) received the PhD degree in computer science from Birla Institute of Technology & Science, Pilani, Rajasthan, India in 2013. He is currently working as Assistant Professor at BITS Pilani. He has actively participated in institutional development activities at BITS Pilani by leading several projects to modernize its IT related activities as well as automate its several academic activities. His research interests include SDN, programmable networks, IoT, edge and cloud computing.

Ashutosh Bhatia is an Assistant Professor in the Department of Computer Science and Information Systems, Birla Institute of Technology and Science Pilani. He completed his PhD and postgraduate degree from Indian Institute of Science (IISc), Bangalore and B.E. degree from Barkatullah University, Bhopal. Earlier, he also worked as scientist in Defence Research and Development Organization (DRDO), India for nearly five years and worked as research engineer in Samsung India. During his stay in Samsung, he was also the member of MAC core committee of IEEE 802.15.6 standardization group, and also, patented a couple of methods related to the MAC protocol of wireless body area network (WBAN). His research interests include building new designs, protocols, algorithms and theories that improve the security, performance and robustness of various networks and systems.