Chapter 5

An Efficient Method to Collect Statistics in SDN Using Curvature Based Sampling

5.1 Introduction

Network Monitoring is a fundamental task in network management. It consists of services that carry out various entities' statistics collection in a network such as flows, devices, ports, and links. Many important applications such as traffic engineering, dynamic routing, network billing, anomaly detection, load balancing, Quality of Services (QoS) management, and Service Level Agreement (SLA) enforcement heavily rely on these services.

In Chapter 3, GlobeSnap collects consistent statistics and in Chapter 4, qMon measures the link delay. In both the chapters, the controller polls the network statistics at a constant rate. They do not provide any mechanism to decide the polling rate. In this chapter we

- Sandhya Rathee, Divesh Uttamchandani, K Haribabu, Ashutosh Bhatia. *An Efficient Method to Collect Statistics in SDN Using Curvature Based Sampling*, In 2021 International Conference on Information Networking (ICOIN), pp. 175-180. IEEE, 2021..

propose a method that adjusts the polling rate dynamically based on the change in the network traffic.

The SDN controller can receive the flow statistics without any additional overhead with the help of PacketIn and FlowRemoved messages from the underlying switches [57]. We call this *push- based* approach. In push-based approach, the controller does not send any request message to the switches to collect the statistics. Instead, it uses control messages or port mirroring [56] to get the network statistics. The problem with push-based approach is that the controller has to wait for all the active flows to expire because the controller can perform the measurements at a given time only after receiving the *FlowRemoved* messages for all the active flows at that time. With port mirroring the issue is that, if the mirrored traffic exceeds the port's capacity then the packets start dropping and can provide wrong measurements.

Another approach is *pull-based* approach, where the controller sends the statistics request messages to the network devices, and devices send the corresponding statistics reply to the controller. Polling in pull-based approach can be performed in two ways, fixed rate polling, e.g., OpenTM [55] and dynamic rate polling, e.g., CeMon [2]. In fixed rate polling, the accuracy of the collected statistics depends on the polling interval. Polling at a higher rate can provide a better measurement of the traffic but can have a large overhead. Whereas, polling at a lower rate gives less overhead but may give inaccurate measurements. Constant rate polling does not take into account the behaviour of the network traffic. Whereas, dynamic rate polling adjusts the polling interval based on the behaviour of the network traffic.

The existing *pull-based* dynamic rate polling methods use the change in the arrival rate of the traffic as the metric to increase or decrease the polling rate [2, 4]. This approach incurs higher overhead, especially when the change in measured/collected statistics is linear. In this chapter, we propose a novel *pull-based* dynamic polling method using curvature based sampling [163]. Additionally, instead of using fixed values for the parameters, we tune them to achieve optimal results. Finally, we also provide a cost function to rank different *pull-based* methods of statistics collection.

5.2 Related Work

In literature, there exist many methods for network monitoring. The effectiveness of these methods varies a lot in terms of cost, accuracy, and promptness. The breadth in these methods is due to the diversity in the network architecture. Methods like sFlow[15] and NetFlow[14] work in traditional networks for statistics collection. These methods require installation of monitoring modules on the network devices, which incurs a considerable cost in terms of time and effort required on the part of network administrator.

Two of the existing *push-based* approaches are FlowSense [57] and Planck [56]. FlowSense [57], computes the link utilization using the information provided by *PacketIn* and *FlowRemoved* messages [32]. The measurements are not timely, as the link utilization calculation at a given time requires that all the active flows at that time should expire and send a FlowRemoved message to the controller. Moreover, the waiting time can vary depending on the flow size. In Planck [56], traffic going through all the ports of a switch is mirrored to one monitoring port, which is connected to a system called collector. The collector does the job of analysing the statistics. The problem with this solution is that the switch starts dropping packets if the traffic volume exceeds the mirrored port's maximum capacity.

Now we discuss some of the existing works which come under *pull-based* approach. OpenTM[55] is one of the initial works in the domain of network monitoring in SDN which uses a fixed polling interval. It calculates the traffic matrix by polling the end switches of a flow. Although polling the end switches for each flow is better in terms of accuracy but can have more overhead on the end switches. To distribute the load evenly, it suggests random polling and round-robin polling methods. Payless[127] proposes a high-level RESTful stats collection API. It maintains a table of active flows along with timeout of τ ms. If the flow expires in τ ms, the controller gets the statistics in *FlowRemoved* message, else after τ ms, the controller sends a flow statistics request message to collect the statistics. They use a threshold of 100 MB to adjust the polling rate. In OpenSample [128], the authors propose a mechanism which uses sFlow [15] to estimate the flow rate and link utilization. However, the estimations provided by their method are dependent on the TCP packet header. It uses the sequence number field of TCP to estimate the flow rate by taking the difference of two sequence numbers in a given sampling interval.

CeMon [2] and MoMon [4] are *pull-based* methods, which provide solution for adaptive polling. CeMon [2] maintains a window of historical stats and adjusts the window size based on the change in the network traffic. The window keeps track of the network history of a flow. When the traffic deviates significantly from the history (more than mean plus twice of standard deviation), the algorithm decreases the window size by half to discount the pass and also doubles the polling frequency. If the traffic is stable, then the polling rate is decreased to half, and the window size is increased by one so that it gives more weight to history. In [4], the authors propose a new method, MoMon [1], to decide the polling frequency in SDN network. Unlike CeMon [2], it uses the recent two polls data to adjust the polling rate [2].

CeMon [2] and MoMon [4] both use fixed values for the factor by which they vary the polling frequency. In MoMon [4], the authors consider 20% change as a significant change. However, the authors did not mention the rationale behind choosing these values.

5.3 Proposed Solution

We propose a new *pull-based* method that adjusts the polling rate depending on the change in the curve of the polled data. The problem of polling can be formulated as that of sampling of a continuous curve into discrete points.

[163] highlights some popular sampling methods for curves. These include,

1. **Arc length sampling**, samples the curve at equal distances on the arc.
2. **Curvature based sampling**, samples the curve in proportion to the instantaneous curvature of the curve.
3. **Mixed sampling**, uses a combination of both arc-based sampling and curvature based sampling.

We cannot use the highlighted methods as it is for sampling network statistics because arc-based sampling assumes the whole curve is available to find and divide the length of the curve into equal parts. This is not possible in our case, as the aim is to get the

[1]we are naming it as **MoMon** for ease of reference.
[2]we use polling rate and polling frequency interchangeably.

network statistics based on recently polled statistics. Also, curvature based sampling needs instantaneous curvature to determine the next discrete point [163], which is not available in case of network traffic. Thus, we substitute it with the difference of the recent two average slopes of the metric (we call it curvature approximation).

The state-of-the-art methods MoMon [4], and CeMon [2], use the criteria "If the change in measured statistics between two consecutive polls is more than some threshold" then increase the polling frequency otherwise decrease the polling frequency. They consider the linear increase and decrease in the statistics to dynamically adjust the polling rate. However, in the proposed method, we use the criteria "If significant curvature is detected between the polls then increase the polling frequency else decrease the polling frequency". We approximate the curvature by considering the percentage change between the slopes within three consecutive polls. In other words, the existing methods use "rate-of-change" in measured metric to decide the polling frequency. Whereas, we consider "change in rate-of-change" as the basis to adjust the polling frequency.

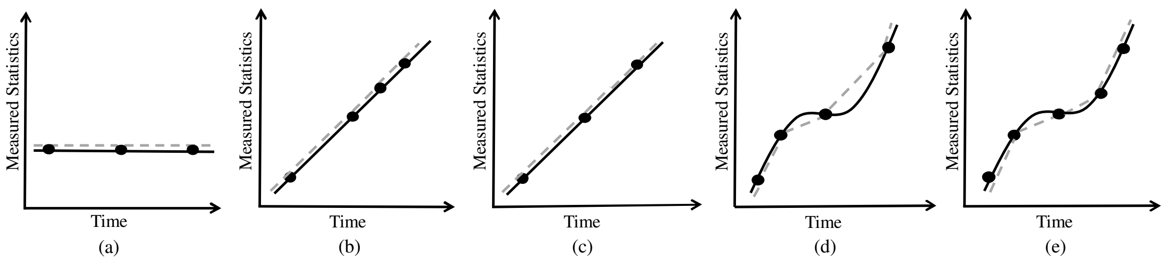

Figure 5.1: (a) shows constant curve with possible polls by CeMon [2], MoMon [4], and the proposed method. The filled circles represent the polled points and dotted lines represent the sampled curve. (b) shows a linearly increasing curve with possible polls by CeMon [2] and MoMon [4]. (c) shows a linearly increasing curve with possible polls by the proposed method. (d) shows a non-linear curve with possible polls by CeMon [2] and MoMon [4]. (e) shows a non-linear curve with possible polls by the proposed method.

Consider Figure 5.1 (a), where we have a constant curve for the measured statistics. That is, there is no change in the measured statistics. The polled points are shown with filled circles on the curve, and the sampled curve is shown by dotted grey line. For constant curve all three methods CeMon [2], MoMon [4], and the proposed method decrease the polling frequency. For a linearly increasing curve as shown in Figure 5.1 (b), CeMon [2] and MoMon [4] compares the difference between the measured statistics at recent two points with a threshold. Based on the comparison, they either increase or decrease the polling rate. The best strategy in such a case is to reduce the polling frequency. As the change in the measured statistics is linear, reducing the polling frequency does not affect the accuracy of the measured statistics and at the same time reduces the polling overhead. The proposed method captures the “change in rate-of-change" by taking the percentage change between two consecutive slopes over a curve. As the slope is constant for a linear curve and percentage change in slopes is zero. Thus for the case shown in Figure 5.1 (b), the proposed method decreases its polling frequency and provides the same accuracy with less overhead as shown in Figure 5.1 (c). Now consider the non-linear curve given in figures 5.1 (d) and (e). The existing methods may or may not change the polling frequency as they consider the difference between the last two poll’s data and thus would not get a good approximation of the actual curve, as shown in Figure 5.1 (d). Whereas, our method increases the polling frequency if the slope of the curve changes by a significant amount (i.e., more than Δ). This might increase the overhead but provides a good approximation of the curve, as shown in Figure 5.1 (e).

5.3.1 Algorithm

The proposed algorithm has five parameters, m_t, m_T, Δ, t_i, and t_d. The polling interval is bounded by m_t and m_T, where m_t, and m_T are the minimum and maximum polling interval time respectively. The value of m_t is taken as 0.5 sec (the same value is used by CeMon [2] and MoMon [4]). The value of m_T is either 3 sec (from MoMon [4]) or 5 sec (from CeMon [2]). The value of other three parameters, Δ, t_i, and t_d, are tuned over real traffic. t_i, and t_d are the factors by which we increase or decrease the polling intervals, respectively. Δ is the threshold. We compare the percentage change in slope with Δ to adjust the polling frequency.

Initially, the polling interval is set to 1 second (line 1 of Algorithm 5.1). We need at least three points to determine the slopes. A window is maintained to store the statistics of last three polls (line 5-6 of Algorithm 5.1). We take the last three poll’s data and calculate the slopes between two consecutive polls. $slope_1$ is calculated between the first and the second polled point, and $slope_2$ is calculated between the second and the third polled point (line 7 of Algorithm 5.1). We compare the percentage change between $slope_1$ and $slope_2$ (which is an approximation of the curvature) with Δ. If the percentage change is

Algorithm 5.1: Curvature Based Sampling

```
Input      : stat, stat_time // Polled stat and time of polling
Parameters: $\Delta$, $t_i$, $t_d$, $m_t$, $m_T$
Output     : Next polling time
// Set the initial values
polling_interval = 1 sec // initial polling interval
win_time ← [];
// time at which data was polled
win_stat ← [];
// stats data
win_dstat ← [];
// slope between two consecutive stats
win_time.append(time);
win_stat.append(stat);
win_dstat.append((win_stat[n]-win_stat[n-1])/(win_time[n]-win_time[n-1]));
// append rate of consecutive elements
if $win\_stat.length > 3$ then
    // only if sufficient entries in window
    if $abs((win\_dstat[n]/win\_dstat[n-1]) - 1) > \Delta$ then
        $polling\_interval = \frac{polling\_interval}{t_d}$;
    else
        $polling\_interval = polling\_interval * t_i$;
    end
    polling_interval ← $max(min(m_T, polling\_interval), m_t)$;
    // bring in range $m_t : m_T$, where $m_t$ is minimum polling
       interval, $m_T$ is maximum polling interval
end
SetTimeout(polling_interval, Curvature Based Sampling) // call the same
    function after the timeout
```

more than Δ then we decrease the polling interval by a factor of t_d (line 10 of Algorithm 5.1), which increases the polling frequency. Otherwise, the polling interval is increased by a factor of t_i (line 12 of Algorithm 5.1).

5.3.2 Cost Function

We compare our method with existing methods CeMon [2], and MoMon [4], in terms of accuracy achieved and overhead incurred. Accuracy is measured using Normalized Root-Mean-Square Error (NRMSE). That is,

$$NRMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}((measured_i - actual_i)/(actual_i))^2} \qquad (5.1)$$

Control message count is used as a measurement of overhead. Cost function is defined as,

$$Cost = NRMSE * Overhead \tag{5.2}$$

We use the cost function for our evaluations and compare the methods by the ratio of the costs.

5.4 Experiments and Evaluation

For the experiments we use the same dataset (given in [164]) as used in CeMon [2] and MoMon [4]. Emulation of large networks becomes infeasible because of the large resource requirements. Thus, we conduct the experiments in a simulator designed in python (which is similar to CeMon [2] and MoMon [4]). The experiments are performed on a machine with i5 processor 2.1GHz CPU processor and 6GB RAM, ubuntu 16.04 OS, python3.7.

For parameter tuning we use a subset of the traffic from the dataset given in [164]. We compare our algorithm's result with CeMon [2] and MoMon [4] for average cost. Initially, we perform coarse tuning of parameters, t_i, t_d, and Δ, by varying the value of each parameter at an interval of 0.5. After coarse tuning we perform fine tuning on best parameter's values found during coarse tuning by varying the values at an interval of 0.1. While tuning t_{min} is always taken as 0.5 sec and t_{max} is either set to 5 sec (from CeMon [2]) or 3 sec (from MoMon [4]). The final values of parameters obtained after tuning are t_{max} = 3.0, Δ=0.5, t_i=2.2, t_d=1.1.

For evaluation, we use link utilization as the underlying metric. We take 60 seconds trace from the data set which is not used in parameter tuning. Cemon [2] polls the network for each flow individually and MoMon [4] does the grouping of flows to reduce the overhead. We have two sets of experiments, first where we implement the Sliding Window Based Tuning (SWT) algorithm (given in CeMon [2]) to poll each flow individually. In the second experiment, CeMon polls the flows in groups. The proposed method and MoMon [4] always poll the flows in groups. Thus, they always has less overhead compared to CeMon [2].

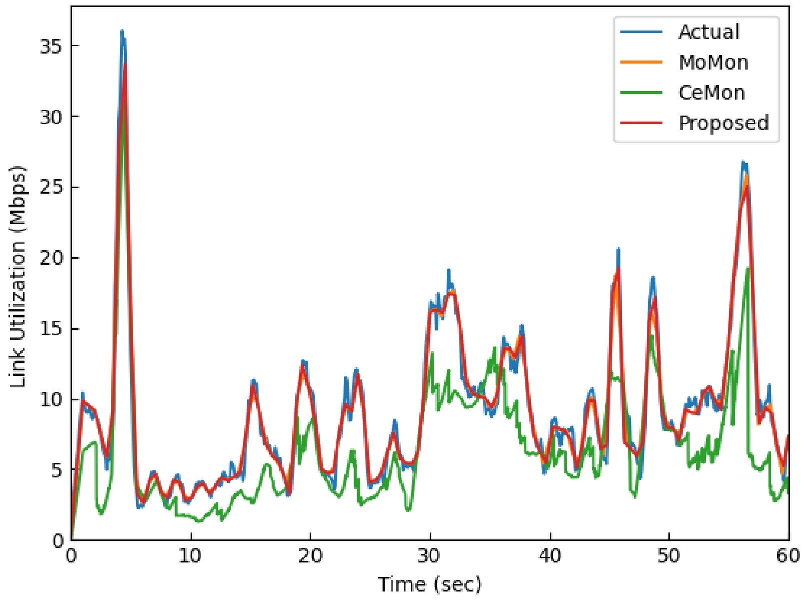

Figure 5.2: Link utilization, CeMon [2] polls the flows individually, and MoMon [4], and proposed method poll the flows in group with same t_{max} value i.e., 3.

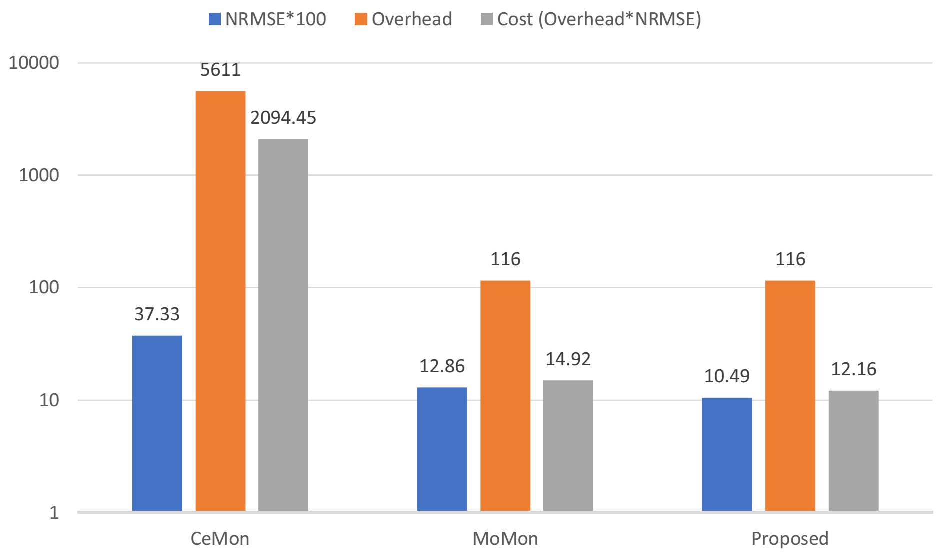

Figure 5.3: Accuracy, overhead, and cost comparison, CeMon [2] polls the flows individually, and MoMon [4], and proposed method poll the flows in group with same t_{max} value i.e., 3.

In both the experiments, we have two different values of t_{max} 3 sec (from MoMon [4]) and 5 sec (from CeMon [2]). t_{min} value is always .5 sec.

1. CeMon [2] polls each flow individually. Whereas, MoMon [4], and the proposed method poll the flows in groups.

- **Case 1:** Same t_{max}

  In this case, we use t_{max} = 3 for all three methods, CeMon [2], MoMon [4], and the proposed method. The estimated link utilization by different methods is given in Figure 5.2 and the accuracy, overhead, and cost comparison is shown in Figure 5.3. The NRMSE value of CeMon [2] is .37 which is roughly 4 times more than the proposed method. Also, the total number of polls done by CeMon [2] are huge i.e., 5611, which results in more cost (approximately 172 times) compared to the proposed method.

- **Case 2:** Different t_{max}

  In this case, we use t_{max} = 5 for CeMon [2], and t_{max} = 3 for MoMon [4] and the proposed method. The estimated link utilization by different methods is given in Figure 5.4 and the accuracy, overhead, and cost comparison is shown in Figure 5.5. The NRMSE value of CeMon [2] is 0.36 which is roughly 3 times more than the proposed method. The total number of polls done by CeMon [2] are 4376, which results in more cost (approximately 128 times) compared to the proposed method.

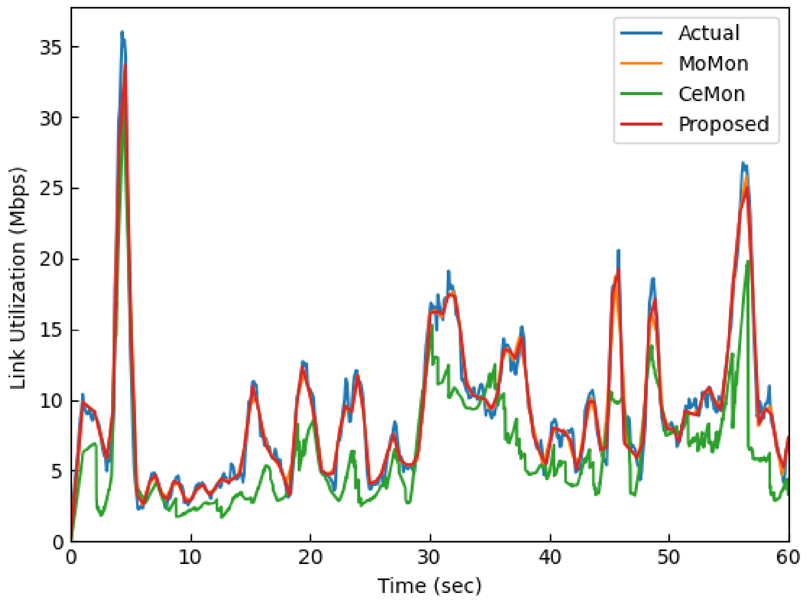

Figure 5.4: Link utilization, CeMon [2] polls the flows individually, and MoMon [4], and proposed method poll the flows in group with different t_{max} values i.e., t_{max} = 5 in case of CeMon and t_{max} = 3 in case of MoMon [4] and proposed method.

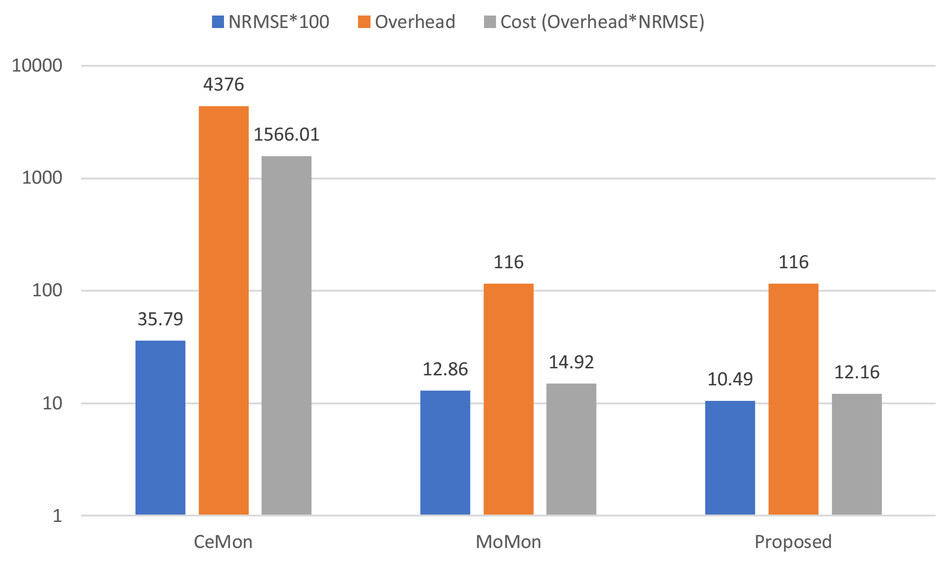

Figure 5.5: Accuracy, overhead, and cost comparison, CeMon [2] polls the flows individually, and MoMon [4], and proposed method poll the flows in group with different t_{max} values i.e., $t_{max} = 5$ in case of CeMon and $t_{max} = 3$ in case of MoMon [4] and proposed method.

In both the cases, the total number of polls for both MoMon [4] and the proposed method are 116 because the tuned value of the parameter t_{max} matches with MoMon [4]. The NRMSE values of MoMon [4] and the proposed method are 0.13 and 0.10, respectively. The cost of MoMon and the proposed method are 14.92 and 12.16, respectively. The proposed method outperforms both CeMon [2] and MoMon [2]. It is better than MoMon [4] by 23% in terms of accuracy and cost and better than CeMon [2] by a huge margin (128 to 172 times) in terms of cost and roughly 3 to 4 times better in terms of accuracy.

2. All the three methods, CeMon [2], MoMon [4], and the proposed method, poll flows in groups.

   - **Case 1:** Same t_{max}

     In this case, we use $t_{max} = 3$ for all three approaches, CeMon [2], MoMon [4], and the proposed method. The link utilization estimated by different methods is given in Figure 5.6 and the accuracy, overhead, and cost comparison is shown in Figure 5.7. The total number of polls done by CeMon [2] are 27 which is roughly 4 times less compared to the proposed method. But CeMon [2] misses lots of spikes (see Figure 5.6), which results in less accuracy. The NRMSE of CeMon [2] is .48 which results in 6.4% more cost compared to the proposed

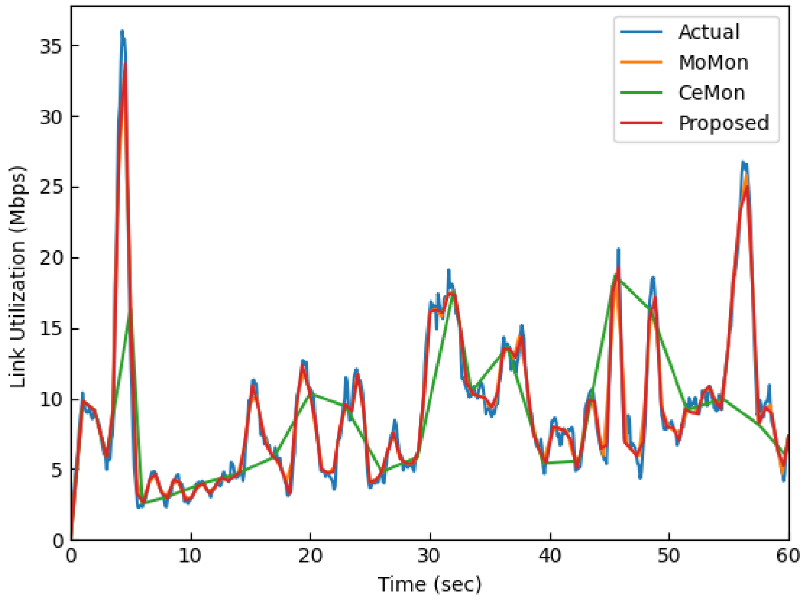

Figure 5.6: Link utilization, CeMon [2], MoMon [4], and proposed method poll the flows in group with same t_{max} value i.e., 3.

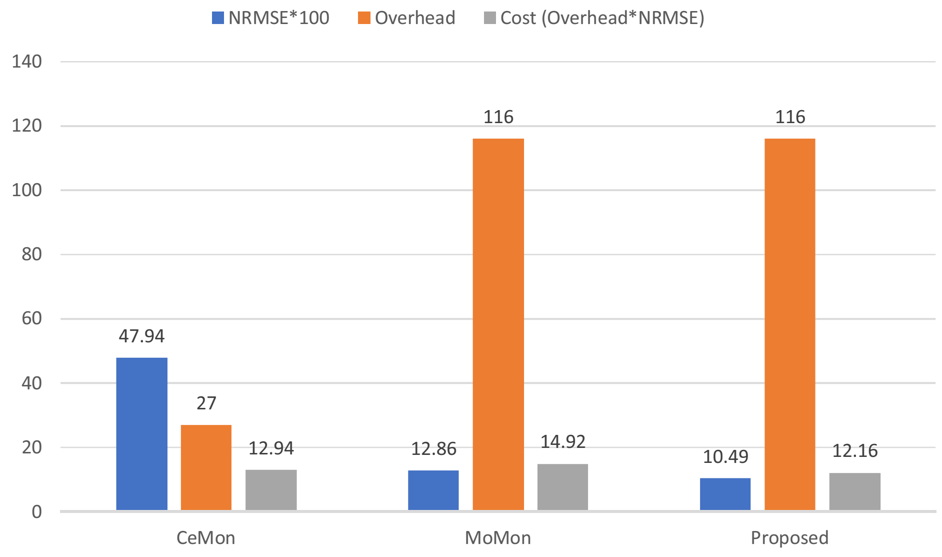

Figure 5.7: Accuracy, overhead, and cost comparison, CeMon [2], MoMon [4], and proposed method poll the flows in group with same t_{max} value i.e., 3.

method.

- **Case 2:** Different t_{max}

  In this case, we use $t_{max} = 5$ for CeMon [2], and $t_{max} = 3$ for MoMon [4] and the proposed method. The link utilization estimated by different methods is given in Figure 5.8 and the accuracy, overhead, and cost comparison is shown

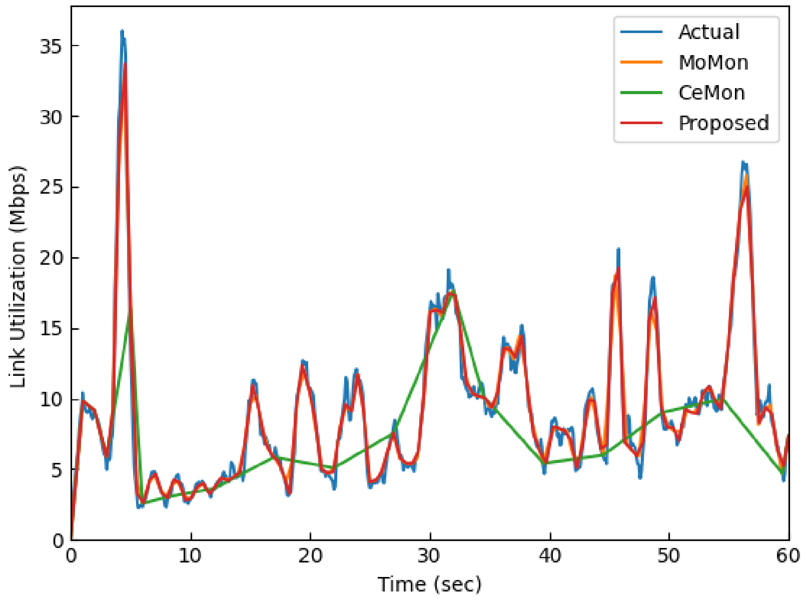

Figure 5.8: Link utilization, CeMon [2], MoMon [4], and proposed method poll the flows in group with different t_{max} values i.e., t_{max} = 5 in case of CeMon and t_{max} = 3 in case of MoMon [4] and proposed method.

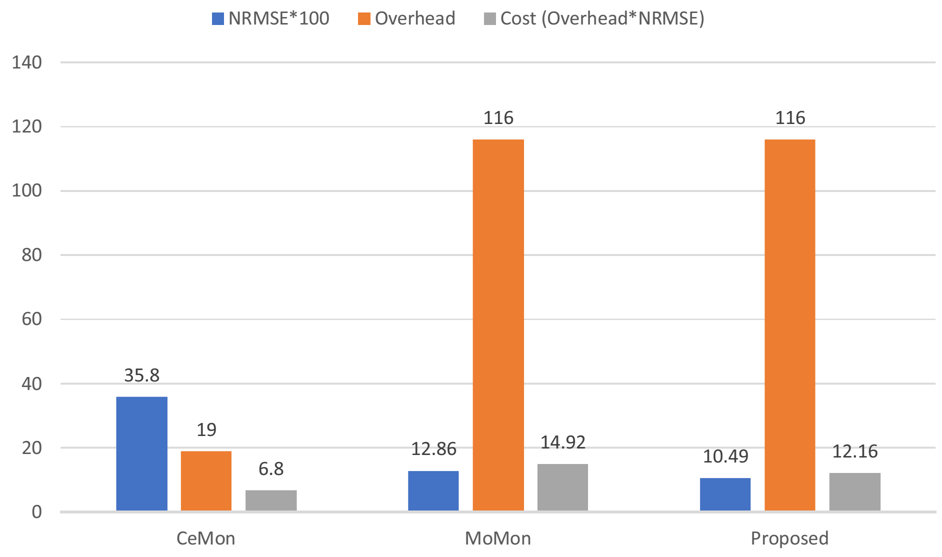

Figure 5.9: Accuracy, overhead, and cost comparison, CeMon [2], MoMon [4], and proposed method poll the flows in group with different t_{max} values i.e., t_{max} = 5 in case of CeMon and t_{max} = 3 in case of MoMon [4] and proposed method.

in Figure 5.9. The total number of polls done by CeMon [2] are 19 which is roughly 6 times less compared to the proposed method. But CeMon [2] misses lots of spikes (see Figure 5.8), which results in less accuracy. The NRMSE of CeMon [2] is 0.36 which is roughly 3 time inaccurate compared to the proposed method.

In both the cases, the total number of polls for both MoMon [4] and the proposed method are 116. The NRMSE of MoMon and the proposed method is 0.13 and 0.10, respectively. The cost of MoMon and the proposed method is 14.92 and 12.16, respectively. We can see that the proposed method gives better results compared to both CeMon [2] and MoMon [4] in terms of accuracy. We are better than MoMon [4] by 23% in terms of accuracy and cost. We are better than CeMon [2] by 6.4% in terms of cost when we use same t_{max} value. Although CeMon [2] is better (roughly 2 times) than the proposed method in terms of cost when we use different t_{max} values. But in both the cases, the proposed method provides better accuracy (roughly 3 to 4 times) over CeMon [2].

5.5 Summary

In this chapter, we proposed a novel pull-based method to determine the polling frequency with which an SDN controller should poll the switches to collect the statistics. The proposed method considers the change in rate-of-change in the collected statistics to adjust the polling frequency. Instead of using the fixed values for the parameters, we tuned the parameters used in the proposed algorithm. We also provide a cost function to evaluate the performance of different methods. The experiments show that the proposed method is better than MoMon [4] by 23% in terms of accuracy and cost. When CeMon [2] polls each flow individually, the proposed method is better than CeMon [2] roughly by 3 to 4 times in terms of accuracy and 128 to 172 times in terms of cost. When CeMon [2] polls the flows in groups with same t_{max}, the proposed method is better by 6.4% in terms of cost. Although CeMon is better than the proposed method in terms of overhead when it polls the flows in groups with different t_{max}. But it misses lots of spikes in the traffic thus falls behind the proposed method in terms of accuracy.