PayLess: A Low Cost Network Monitoring Framework for Software Defined Networks

Shihabur Rahman Chowdhury, Md. Faizul Bari, Reaz Ahmed, and Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo
{sr2chowdhury | mfbari | r5ahmed | rboutaba}@uwaterloo.ca

***Abstract*—Software Defined Networking promises to simplify network management tasks by separating the control plane (a central controller) from the data plane (switches). OpenFlow has emerged as the *de facto* standard for communication between the controller and switches. Apart from providing flow control and communication interfaces, OpenFlow provides a flow level statistics collection mechanism from the data plane. It exposes a high level interface for per flow and aggregate statistics collection. Network applications can use this high level interface to monitor network status without being concerned about the low level details. In order to keep the switch design simple, this statistics collection mechanism is implemented as a pull-based service, i.e. network applications and in turn the controller has to periodically query the switches about flow statistics. The frequency of polling the switches determines monitoring accuracy and network overhead. In this paper, we focus on this trade-off between monitoring accuracy, timeliness and network overhead. We propose PayLess – a monitoring framework for SDN. PayLess provides a flexible RESTful API for flow statistics collection at different aggregation levels. It uses an adaptive statistics collection algorithm that delivers highly accurate information in real-time without incurring significant network overhead. We utilize the Floodlight controller's API to implement the proposed monitoring framework. The effectiveness of our solution is demonstrated through emulations in Mininet.**

I. Introduction

Monitoring is crucial to network management. Management applications require accurate and timely statistics on network resources at different aggregation levels. Yet, the network overhead for statistics collection should be minimal. Accurate and timely statistics is essential for many network management tasks, like load balancing, traffic engineering, enforcing Service Level Agreement (SLA), accounting and intrusion detection. Management applications may need to monitor network resources at different aggregation levels. For example, an ISP's billing system would require monthly upstream and downstream usage data for each user, an SLA enforcement application may require per queue packet drop rate at ingress and egress switches to ensure bounds on packet drops, a load balancing application may require a switch's per port traffic per unit time.

A well designed network monitoring framework should provide the management applications with a wide selection of network metrics to monitor at different levels of aggregation, accuracy and timeliness. Ideally, it is the responsibility of the monitoring framework to select and poll the network resources unless otherwise specified by the management applications. The monitoring framework should accumulate, process and deliver the monitored data at requested aggregation level and frequency, without introducing too much monitoring overhead into the system.

Although accurate and timely monitoring is essential for seamless network management, contemporary solutions for monitoring IP networks are ad-hoc in nature and hard to implement. Monitoring methods in IP networks can be classified as direct and sampling based [1], [2], [3]. Direct measurement methods incur significant network overhead, while sampling based methods overcome this problem by sacrificing accuracy. Moreover, different network equipment vendors have proprietary technologies to collect statistics about the traffic [1], [3], [4]. The lack of openness and interoperability between these methods and technologies have made the traffic statistics collection a complex task in traditional IP networks.

More recently, *Software Defined Networking (SDN)* has emerged with the promise to facilitate network programmability and ease the management tasks. SDN proposes to decouple control plane from data plane. Data plane functionality of packet forwarding is built into switching fabric, whereas the control plane functionality of controlling network devices is placed in a logically centralized software component called controller. The control plane provides a programmatic interface for developing management programs, as opposed to providing a configuration interface for tuning network properties. From a management point of view, this added programmability opens the opportunity to reduce the complexity of distributed configuration and ease the network management tasks [5].

The OpenFlow [6] protocol has been accepted as the *de facto* interface between the control and data planes. OpenFlow provides per flow[1] statistics collection primitives at the controller. The controller can poll a switch to collect statistics on the active flows. Alternatively, it can request a switch to push flow statistics (upon flow timeout) at a specific frequency. The controller has a global view of the network. Sophisticated and effective monitoring solutions can be developed using these capabilities of an OpenFlow Controller. However, in the current scenario, a network management application for SDN, would be a part of the control plane, rather than being independent of it. This is due to the heterogeneity in the controller technologies, and the absence of a uniform abstract view of the network resources.

In this paper, we propose *PayLess*, a network monitoring framework for SDN. PayLess offers a number of advantages

[1] A *flow* is identified by a ordered set of Layer 2-4 header fields

978-1-4799-0913-1/14/$31.00 © 2014 IEEE

towards developing network management applications on top of the SDN controller platform. First, PayLess provides an abstract view of the network and an uniform way to request statistics about the resources. Second, PayLess itself is developed as a collection of pluggable components. Interaction between these components are abstracted by well-defined interfaces. Hence, one can develop custom components and plug into the PayLess framework. Highly variable tasks, like data aggregation level and sampling method, can be easily customized in PayLess. We also study the resource-accuracy trade-off issue in network monitoring and propose a variable frequency adaptive statistics collection scheduling algorithm.

The rest of this paper is organized as follows. We begin with a discussion of some existing IP network monitoring tools, OpenFlow inspired monitoring tools, and variable rate adaptive data collection methods used in sensor and IP networks (Section II). Then we present the architecture of PayLess (Section III) followed by a presentation of our proposed flow statistics collection scheduling algorithm (Section IV). The next section describes the implementation of a link utilization monitoring application using the proposed algorithm (Section V). We evaluate and compare the performance of our link utilization monitoring application with that of FlowSense [7] through simulations using Mininet (Section VI). Finally, we conclude this paper and point out some future directions of our work (Section VII).

II. Related Works

There exists a number of flow based network monitoring tools for traditional IP networks. NetFlow [1] from Cisco is the most prevalent one. NetFlow probes are attached to a switch as special modules. These probes collect either complete or sampled traffic statistics, and send them to a central collector [4]. NetFlow version 9 has been adopted to be a common and universal standard by IP Flow Information Export (IPFIX) IETF working group, so that non-Cisco devices can send data to NetFlow collectors. NetFlow provides information such as source and destination IP address, port number, byte count, *etc*. It supports different technologies like multi-cast, IPSec, and MPLS. Another flow sampling method is sFlow [2], which was introduced and maintained by InMon as an open standard. It uses time-based sampling for capturing traffic information. Another proprietary flow sampling method is JFlow [3], developed by the Juniper Networks. JFlow is quite similar to NetFlow. JFlow provides detailed information about each flow by applying statistical sampling just like NetFlow and sFlow. Except for sFlow, NetFlow and JFlow are both proprietary solutions and incur a large up-front licensing and setup cost to be deployed in a network. sFlow is less expensive to deploy, but it is not widely adopted by the vendors.

Recently a good number of network monitoring tools based on OpenFlow have been proposed. Yu *et al.* proposes OpenSketch [8], a three stage packet processing pipeline design for SDN. Opensketch allows the development of more expressive traffic measurement application by proposing a clean slate design of the packet processing pipeline. OpenTM [9] focuses more on efficiently measuring traffic matrix using existing technology. It proposes several heuristics to choose an optimal set of switches to be monitored for each flow. After a switch has been selected it is continuously polled for collecting flow level statistics. Instead of continuously polling a switch, PayLess offers an adaptive scheduling algorithm for polling that achieves the same level of accuracy as continuous polling with much less communication overhead. In [10] the authors have motivated the importance of identifying large traffic aggregates in a network and proposed a monitoring framework utilizing secondary controllers to identify and monitor such aggregates using a small set of rules that changes dynamically with traffic load. This work differs significantly from PayLess. Whereas PayLess's target is to monitor all flows in a network, this work monitors only large aggregate flows. FlowSense [7] proposes a passive push based monitoring method where `FlowRemoved` messages are used to estimate per flow link utilization. While communication overhead for FlowSense is quite low, its estimation is quite far from the actual value and it works well only when there is a large number of small duration flows. FlowSense cannot capture traffic bursts if they do not coincide with another flow's expiry. More recently authors in [11] have proposed an adaptive SDN based monitoring method, focusing on anomaly detection. They manipulate the aggregation level of the packet forwarding rules to switch between different granularity levels of traffic measurement. In contrast, we propose a framework for developing a wide range of network monitoring application and a variable rate adaptive sampling algorithm, which makes an adaption in the time scale.

There has been an everlasting trade-off between statistics collection accuracy and resource usage for monitoring in IP networks. Monitoring in SDN also needs to make a trade-off between resource overhead and measurement accuracy as discussed by the authors in [12]. Variable rate adaptive sampling techniques have been proposed in different contexts to improve the resource consumption while providing satisfactory levels of accuracy of collected data. Variable rate sampling techniques to save resource while achieving a higher accuracy rate have been extensively discussed in the literature in the context sensor networks [13], [14], [15], [16], [17], [18]. The main focus of these sampling techniques has been to effectively collect data using the sensor while trying to minimize the sensor's energy consumption, which is often a scarce resource for the sensors. Adaptive sampling techniques have also been studied in the context of traditional IP networks [19], [20]. However, to the best of our knowledge adaptive sampling for monitoring SDN have not been explored yet.

III. System Description

A. PayLess Architecture

Fig. 1 shows the software stack for a typical SDN setup along with our monitoring framework. OpenFlow controllers (*e.g.*, NOX [21], POX [22], Floodlight [23], *etc.*) provide a platform to write custom network applications that are oblivious to the complexity and heterogeneity of the underlying network. An OpenFlow controller provides a programming interface, usually refereed to as the *Northbound API*, to the network applications. Network applications can obtain an abstract view of the network through this API. It also provides interfaces for controlling traffic flows and collecting statistics at different aggregation levels (*e.g.*, flow, packet, port, *etc.*). The required statistics collection granularity varies from application to application. Some applications require per flow statistics, while for others, aggregate statistics is required.

For example, an ISP's user billing application would expect to get usage data for all traffic passing though the user's home router. Unfortunately, neither the OpenFlow API nor the available controller implementations (*e.g.*, NOX, POX and Floodlight) support these aggregation levels. Moreover, augmenting a controller's implementation with monitoring functionality will greatly increase design complexity. Hence, a separate layer for abstracting monitoring complexity from the network applications and the controller implementation is required.

To this end, we propose PayLess: a low-cost efficient network statistics collection framework. PayLess is built on top of an OpenFlow controller's northbound API and provides a high-level RESTful API. The monitoring framework takes care of the translation of high level monitoring requirements expressed by the applications. It also hides the details of statistics collection and storage management. The network monitoring applications, built on top of this framework, will use the RESTful API provided by PayLess and will remain shielded from the underlying low-level details.

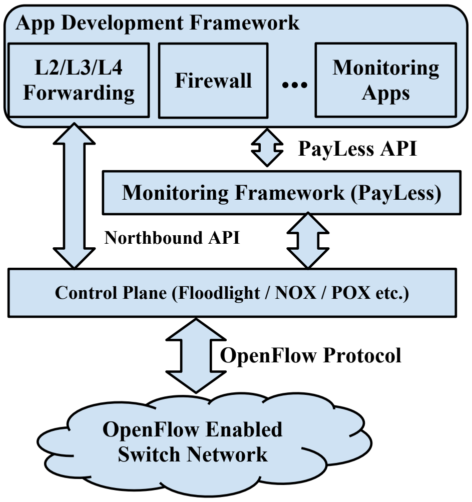
App Development Framework
L2/L3/L4 Forwarding
Firewall
...
Monitoring Apps
PayLess API
Monitoring Framework (PayLess)
Northbound API
Control Plane (Floodlight / NOX / POX etc.)
OpenFlow Protocol
OpenFlow Enabled Switch Network

Fig. 1. SDN Software Stack

We elaborate the monitoring framework (PayLess) portion from Fig. 1 and show its components in Fig. 2. These component are explained in detail below:

- **Request Interpreter**: This component is responsible for translating the high level primitives expressed by the applications to flow level primitives. For example, a user billing application may request usage of a user by specifying the user's identity (*e.g.*, email address or registration number). This component is responsible for interacting with other modules to translate this high level identifier to network level primitives.
- **Scheduler**: The scheduler component schedules polling of switches in the network for gathering statistics. OpenFlow enabled switches can provide per flow statistics, per queue statistics as well as per port aggregate statistics. The scheduler determines which type of statistics to poll, based on the nature of request it received from an application. The time-stamps of polling is determined by a scheduling algorithm. In the next section, we describe a statistics collection scheduling algorithm for our framework. However, the scheduler is well isolated from the other components in our framework. One can develop customized scheduling algorithm for statistics collection and seamlessly integrate withing the PayLess framework.
- **Switch Selector**: We have to identify and select one (or more) switches for statistics collection, when a statistics collection event is scheduled. This component determines the set of switches to poll for obtaining the required statistics at the schedules time stamps. For example, to collect statistics about a flow, it is sufficient to query the ingress switch only, and it is possible to determine the statistics for the intermediate switches by simple calculations. Authors in [9] have discussed a number of heuristics for switch selection in the context of traffic matrix calculation in SDN.
- **Aggregator & Data Store**: This module is responsible for collecting raw data from the selected switches and storing these raw data in the data store. This module aggregates the collected raw-data to compute monitoring information at requested aggregation levels. The data store is an abstraction of a persistent storage system. It can range from regular files to relational databases to key-value stores.

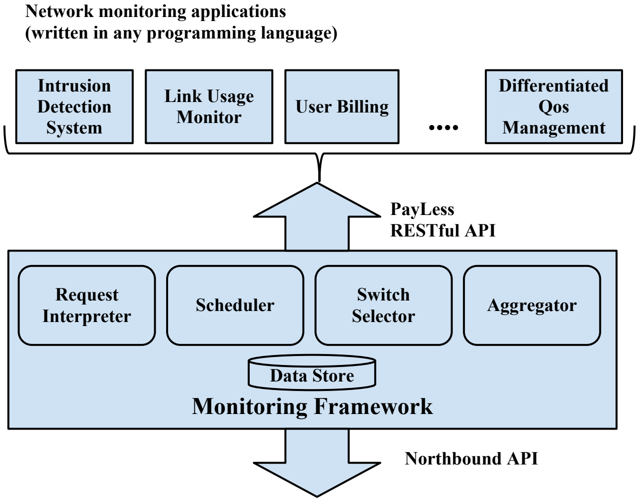

Fig. 2. PayLess Network Monitoring Framework

B. Monitoring API

PayLess provides an RESTful API for rapid development of network monitoring applications. Any programming language can used to access this API. A network application can express high level primitives in its own context to be monitored and get the collected data from the PayLess data store at different aggregation levels. Here, we provide a few examples to illustrate how network applications can access this API. Every network application needs to create a `MonitoringRequest` (Fig. 3) object and register it with PayLess. The `MonitoringRequest` object contains the following information about a monitoring task:

```
{"MonitoringRequest": {
    "Type": "["performance" | "security" | "failure" | ... ]",
    "Metrics": [
        {"performance": ["latency", "jitter", "throughput", "packet-drop", ...]},
        {"security": ["IDS-alerts", "ACL-violations", "Firewall-alerts", ...]},
        {"failure": ["MTBF", "MTTR"]}
    ],
    "Entity": ["<uri_to_script>"],
    "AggregationLevel": ["flow" | "table" | "port" | "switch" | "user" | "custom": "uri_to_script"],
    "Priority": ["real-time", "medium", "low", custom: "monitoring-frequency"],
    "Monitor" : ["direct", "adaptive", "random-sampling", "optimized", "custom": "uri_to_script"],
    "Logging": ["default", "custom": "<uri_to_log_format>"]
}}
```

Fig. 3. MonitoringRequest object

- **Type:** the network application needs to specify what type of metrics it wants to be monitored *e.g.*, performance, security, fault-tolerance, *etc*.
- **Metrics:** for each selected monitoring type, the network application needs to provide the metrics that should be monitored and logged. Performance metrics may include delay, latency, jitter, throughput, *etc*. For security monitoring, metrics may include IDS-alerts, firewall-alerts, ACL-violations *etc*. for a specific switch, port, or user. Failure metrics can be mean-time-before-failure or mean-time-to-repair for a switch, link, or flow table.
- **Entity:** this is an optional parameter and depends on the type of metric to be monitored. This parameter specifies the network entities that need to be monitored. In PayLess, network users[2], switches, switch ports, flow-tables, traffic flows, *etc*. can be uniquely identified and monitored. Network monitoring application can specify which entities it wants to monitor or the field can be left empty as well, depending on the type of metric to be monitored.
- **Aggregation Level:** network applications must specify the aggregation level (*e.g.*, flow, port, user, switch *etc*.) for statistics collection. PayLess provides a set of predefined aggregation levels (Fig. 3), as well as the option to provide a script to specify custom aggregation levels.
- **Priority:** PayLess provides the option to set priority levels for each metric to be monitored. We have three pre-defined priority levels: real-time, medium, and low. Alternatively, an application can specify a custom polling frequency. PayLess framework is responsible for selecting the appropriate polling frequencies for the pre-defined priorities.
- **Monitor:** This parameter specifies the monitoring method, for example, direct, adaptive, random-sampling, or optimized. The default monitoring method is optimized, in which case the PayLess framework selects the appropriate monitoring method for balancing between accuracy, timeliness, and network overhead. Apart from the predefined sampling methods, an application may provide a link to a customized monitoring method.
- **Logging:** A network application can optionally provide a `LogFormat` object to the framework for customizing the output format. If no such object is provided then PayLess writes the logs in its default format.

The `MonitoringRequest` object is specified using JSON. Attributes of this object along with some possible values are shown in Fig. 3. A network application registers a `MonitoringRequest` object through PayLess's RESTful API. After the registration is successful, PayLess provisions monitoring resources for capturing the requested statistics and places them in the data store. In response to a monitoring request PayLess returns a data `access-id` to the network application. The network application uses this access-id to retrieve collected data from the data store.

For example, an ISP's network application for user billing may specify the `MonitoringRequest` object as shown in Fig. 4. Here, the application wants to monitoring performance metrics: throughput, and packet-drops for particular users with a low priority using direct monitoring technique and log the collected data in PayLess's default format.

```
{"MonitoringRequest": {
    "Type": "["performance"]",
    "Metrics": [
        {"performance": [
         "throughput",
         "packet-drop",
        ]},
    ],
    "Entity": ["user": "<user_id>"],
    "AggregationLevel": ["user"],
    "Priority": ["medium"],
    "Monitor" : ["direct"],
    "Logging": ["default"]
}}
```

Fig. 4. `MonitoringRequest` for user billing application

Another example will be a real-time media streaming service that needs to provide differentiated QoS to the user. This application needs flow-level real-time monitoring data to make optimal routing decisions. A possible sample for the `MonitoringRequest` object is shown in Fig. 5.

[2] network users can be identified in the way as described in [24]

```
{"MonitoringRequest": {
    "Type": "["performance"]",
    "Metrics": [
        {"performance": [
         "throughput",
         "latency",
         "jitter",
         "packet-drop",
        ]},
    ],
    "Entity": ["flow": "<flow_specification>"],
    "AggregationLevel": ["flow"],
    "Priority": ["real-time"],
    "Monitor" : ["adaptive"],
    "Logging": ["default"]
}}
```

Fig. 5. MonitoringRequest for differentiated QoS

PayLess also provides API functions for listing, updating, and deleting MonitoringRequest objects. Table I provides a few example API URIs and their parameters for illustration purpose. The first URIs provides the basic CRUD functionality for the MonitorRequest object. The fifth URI is used for accessing collected data from the data store.

IV. An Adaptive Monitoring Method

In this section, we present an adaptive monitoring algorithm that can be used to monitor network resources. Our goal is to achieve accurate and timely statistics, while incurring little network overhead. We assume that the underlying switch to controller communication is performed using the OpenFlow protocol. Therefore, before diving into the details of the algorithm, we present a brief overview of the OpenFlow messages that are used in our framework.

OpenFlow identifies a flow using the fields obtained from layer 2, layer 3 and layer 4 headers of a packet. When a switch receives a flow that does not match with any rules in its forwarding table, it sends a PacketIn message to the controller. The controller installs the necessary forwarding rules in the switches by sending a FlowMod message. The controller can specify an idle timeout for a forwarding rule. This refers to the inactivity period, after which a forwarding rule (and eventually the associated flow) is evicted from the switch. When a flow is evicted the switch sends a FlowRemoved message to the controller. This message contains the duration of the flow as well as the number of bytes matching this flow entry in the switch. Flowsense [7] proposes to monitor link utilization in zero cost by tracking the PacketIn and FlowRemoved messages only. However, this method has large average delay between consecutive statistics retrieval. It also does not perform well in monitoring traffic spikes. In addition to these messages, the controller can send a FlowStatisticsRequest message to the switch to query about a specific flow. The switch sends the duration and byte count for that flow in a FlowStatisticsReply message to the controller.

An obvious approach to collect flow statistics is to poll the switches periodically each constant interval of time by sending the FlowStatisticsRequest message. A high frequency (*i.e.*, low polling interval) of polling will generate highly accurate statistics. However, this will induce significant monitoring overhead in the network. To strike a balance between statistics collection accuracy and incurred network overhead, we propose a variable frequency flow statistics collection algorithm.

We propose that when the controller receives a PacketIn message, it will add a new flow entry to an active flow table along with an initial statistics collection timeout, τ milliseconds. If the flow expires within τ milliseconds, the controller will receive its statistics in a FlowRemoved message. Otherwise, in response to the timeout event (*i.e.*, after τ milliseconds), the controller will send a FlowStatisticsRequest message to the corresponding switch to collect statistics about that flow. If the collected data for that flow does not significantly change within this time period, *i.e.*, the difference between the previous and current byte count against that flow is not above a threshold, say Δ_1, the timeout for that flow is multiplied by a small constant, say α. For a flow with low packet rate, this process may be repeated until a maximum timeout value of $\mathcal{T}_{max}$ is reached. On the other hand, if the difference in the old and new data becomes larger than another threshold Δ_2, the scheduling timeout of that flow is divided by another constant β. For a heavy flow, this process may be repeated until a minimum timeout value of $\mathcal{T}_{min}$ is reached. The rationale behind this timeout adjustment is that we maintain a higher polling frequency for flows that significantly contribute to link utilization, and we maintain a lower polling frequency for flows that do not significantly contribute towards link utilization at that moment. If their contribution increases, the scheduling timeout will adjust according to the proposed algorithm to adapt the polling frequency with the increase in traffic.

We optimize this algorithm further by batching FlowStatisticsRequest messages together for flows with same timeout. This will reduce the spread of monitoring traffic in the network without affecting the effectiveness of polling with a variable frequency. The pseudocode of this algorithm is shown in Algorithm 1.

V. Implementation: Link Utilization Monitoring

As a concrete use case of our proposed framework and the monitoring algorithm, we have implemented a prototype link utilization monitoring application on Floodlight controller platform. We have chosen Floodlight as the controller platform for its highly modular design and the rich set of APIs to perform operations on the underlying OpenFlow network.

It is worth mentioning that our prototype implementation is intended to perform experiments and to show the effectiveness of our algorithm. Hence, we have made the following simplifying assumption about flow identification and matching without any loss of generality. Since we are monitoring link utilization, it is sufficient for us to identify the flows by their source and destination IP addresses. We performed the experiments using *iperf* [25] in UDP mode. The underlying network also had some DHCP traffic, which also uses UDP. We filtered out the DHCP traffic while adding the flows to active flow table by looking at the destination UDP port numbers[3]. It is worth

[3] DHCP uses destination port 67 and 68 for DHCP requests and replies, respectively

RESTful API URI	Parameter(s)
/payless/object/monitor_request/register	data=<JSON data as shown in Fig. 3>
/payless/object/monitor_request/update	id=<request_id>&data=<JSON data as shown in Fig. 3>
/payless/object/monitor_request/list	id=<application_id>
/payless/object/monitor_request/delete	id=<request_id>
/payless/log/retrieve	access-id=<access_id>

TABLE I. PayLess RESTful API

Algorithm 1 FlowStatisticsCollectionScheduling(*Event e*)

```
globals:  $active\_flows$ //Currently Active Flows
          $schedule\_table$ //Associative table of active flows
                            // indexed by poll frequency
          $U$ // Utilization Statistics. Output of this algorithm
if $e$ is Initialization event then
    $active\_flows \leftarrow \phi$, $schedule\_table \leftarrow \phi$, $U \leftarrow \phi$
end if
if $e$ is a PacketIn event then
    $f \leftarrow \langle e.switch, e.port, \mathcal{T}_{min}, 0 \rangle$
    $schedule\_table[\mathcal{T}_{min}] \leftarrow schedule\_table[\mathcal{T}_{min}] \cup f$
else if $e$ is timeout $\tau$ in $schedule\_table$ then
    for all flows $f \in schedule\_table[\tau]$ do
        send a FlowStatisticsRequest to $f.switch$
    end for
else if $e$ is a FlowStatisticsReply event for flow $f$
then
    $diff\_byte\_count \leftarrow e.byte\_count - f.byte\_count$
    $diff\_duration \leftarrow e.duration - f.duration$
    $checkpoint \leftarrow current\_time\_stamp$
    $U[f.port][f.switch][checkpoint] \leftarrow \langle diff\_byte\_count,$
                                              $diff\_duration \rangle$
    if $diff\_byte\_count < \Delta_1$ then
        $f.\tau \leftarrow \min(f.\tau\alpha, \mathcal{T}_{max})$
        Move $f$ to $schedule\_table[f.\tau]$
    else if $diff\_byte\_count > \Delta_2$ then
        $f.\tau \leftarrow \max(f.\tau/\beta, \mathcal{T}_{min})$
        Move $f$ to $schedule\_table[f.\tau]$
    end if
end if
```

noting that all the components of our proposed monitoring framework are not in place yet. Therefore, we resorted to implementing the link utilization monitoring application as a floodlight module.

We intercepted the PacketIn and FlowRemoved messages to keep track of flow installations and removals from the switches, respectively. We also maintained a hash table indexed by the schedule timeout value. Each bucket with timeout τ, contains a list of active flows that need to be polled every τ milliseconds. Each of the bucket in the hashtable is also assigned a worker thread that wakes up every τ milliseconds and sends a FlowStatisticsRequest message to the switches corresponding to the flows in its bucket. The FlowStatisticsReply messages are received asynchronously by the monitoring module. The latter creates a measurement checkpoint for each reply message. The contribution of a flow is calculated by dividing its differential byte count from the previous checkpoint by the differential time duration from the previous checkpoint. The monitoring module examines the measurement checkpoints of the corresponding link and updates the utilization at previous checkpoints if necessary. The active flow entries are moved around the hashtable buckets with lower or higher timeout values depending on the change in byte count from previous measurement checkpoint. Currently, we have a basic REST API, which provides an interface to get the link statistics (in JSON format) of all the links in the network. However, our future objective is to provide a REST API for allowing external applications to register a particular flow for monitoring and obtaining the statistics.

Although the current implementation makes some assumption about flow identification and matching, this does not reduce the generality of our proposed algorithm. Our long term goal is to have a full functional implementation of the PayLess framework for efficient flow statistics collection. Developing network monitoring applications will be greatly simplified by the statistics exposed by our framework. It is worth mentioning that the proposed scheduling algorithm lies at the core of the scheduler component of this framework, and no assumption about the algorithm's implementation were made in this prototype. The only assumptions made here corresponds to the implementation of link utilization monitoring application that uses our framework.

VI. Evaluation

In this section, we present the performance of a demo application for monitoring link utilization. This application is developed using the PayLess framework. We have also implemented Flowsense and compared it to PayLess, since both target the same use case. We have also implemented a baseline scenario, where the controller periodically polls the switches at a constant interval to gather link utilization information. We have used Mininet to simulate a network consisting of hosts and OpenFlow switches. Details on the experimental setup is provided in Section VI-A. Section VI-B explains the evaluation metrics. Finally, the results are presented in Section VI-C.

A. Experiment Setup

We have used a 3-level tree topology as shown in Fig. 7 for this evaluation. UDP flows for a total duration of 100s between hosts were generated using *iperf*. Fig. 6 is the timing diagram showing the start time, throughput and the end time for each flow. We have set the idle timeout of the active flows in a switche to 5s. We have also deliberately introduced pauses of different durations between the flows in the traffic to experiment with different scenarios. Pauses less than the soft timeout were placed between 28th and 30th second, and also between 33 and 35 seconds to observe how the proposed scheduling algorithm and the Flowsense react to sudden traffic spikes. The minimum and maximum polling interval for our scheduling algorithm was set to 500ms and 5s, respectively. For the constant polling case, a polling interval of 1s was used. The parameters Δ_1 and Δ_2 described in Section IV were set to 100MB. Finally, we have set α and β described in Section IV

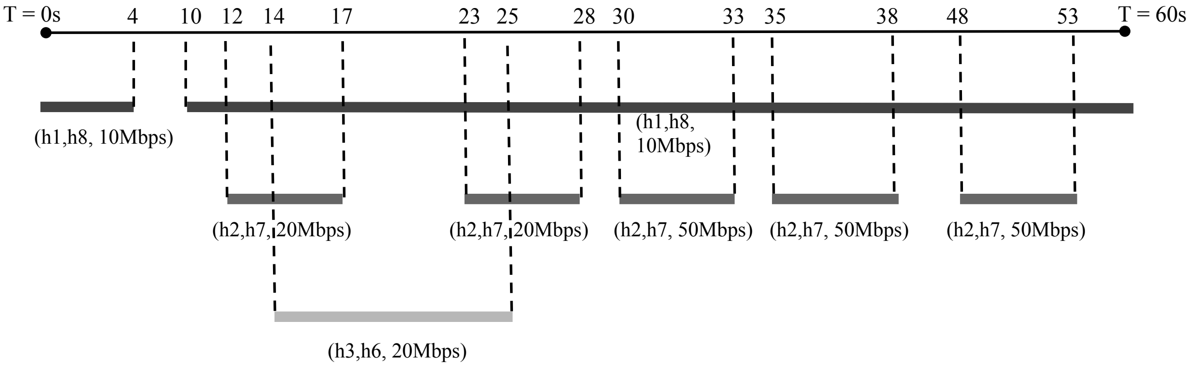

Fig. 6. Timing Diagram of Experiment Traffic

to 2 and 6, respectively. β was set to a higher value to quickly react and adapt to any change in traffic.

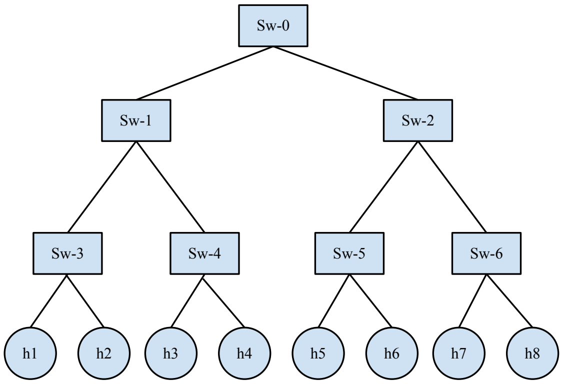

Fig. 7. Topology for Experiment

B. Evaluation Metrics

Utilization: Link utilization is measured as the instantaneous throughput obtained from that link and is measured in units of Mbps. We report the utilization of the link between switches `Sw-0` and `Sw-1` (Fig. 7). According to the traffic mix, this link is part of all the flows and is most heavily used. It also exhibits a good amount of variation in utilization. We also experiment with different values of minimum polling interval ($\mathcal{T}_{min}$) and show its effect on the trade-off between accuracy and monitoring overhead.

Overhead: We compute overhead in terms of the number of `FlowStatisticsRequest` messages sent from the controller. We compute the overhead at timeout expiration events when a number of flows with the same timeout are queried for statistics.

C. Results

1) Utilization: Fig. 8 shows the utilization of `Sw0-Sw1` link over simulation time, measured using three different techniques. The baseline scenario, *i.e.*, periodic polling, which has the most resemblance with the traffic in Fig. 6. Flowsense fails to capture the traffic spikes because of the large granularity of its measurement. The traffic pauses less than the soft timeout value cause Flowsense to report less than the actual utilization. In contrast, our proposed algorithm very closely follows the utilization pattern obtained from periodic polling. Although it did not succeed to fully capture the first spike in the traffic, it quickly adjusted itself to successfully capture the next traffic spike.

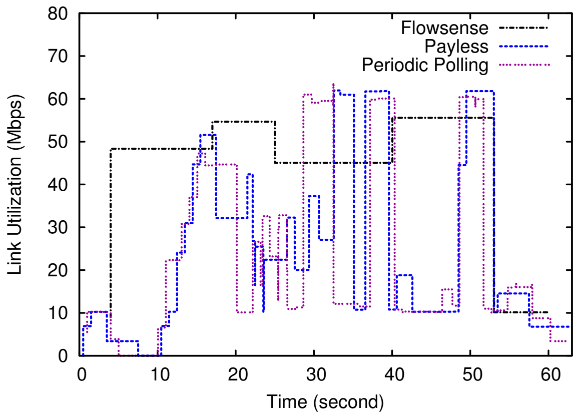

Fig. 8. Utilization Measurement

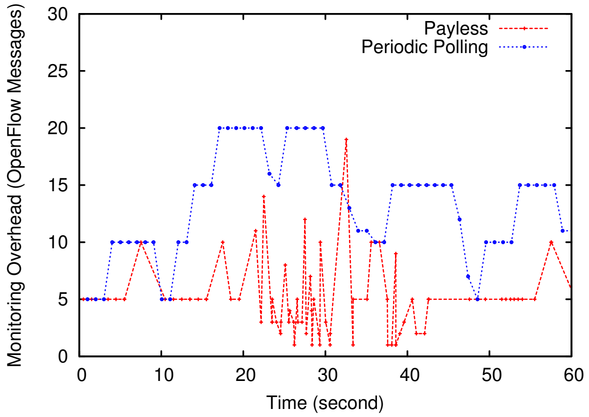

Fig. 9. Messaging Overhead

2) Overhead: Fig. 9 shows the messaging overhead of the baseline scenario and our proposed algorithm. Since Flowsense does not send `FlowStatisticsRequest` messages, therefore it has zero messaging overhead, hence not shown in the figure. The fixed polling method polls all the active flows after the fixed timeout expires. This causes a large number of messages to be injected in the network at the query time. On the other hand, our proposed algorithm reduces the spike of these messages by assigning different timeouts to flows and spreading the messages over time. It is also evident in Fig. 9 that our algorithm has more query points across the timeline, but at each time line it sends out less messages in the network to get statistics about flows. In some cases, our algorithm sends out 50% less messages than that of periodic polling method.

Although Flowsense has zero measurement overhead, it is much less accuracy compared to our adaptive scheduling algorithm. In addition, the monitoring traffic incurred by PayLess is very low, only 6.6 messages per second on average, compared to 13.5 messages per second on average for periodic polling. In summary, the proposed algorithm for scheduling flow statistics can achieve an accuracy close to constant periodic polling method, while having a reduced messaging overhead.

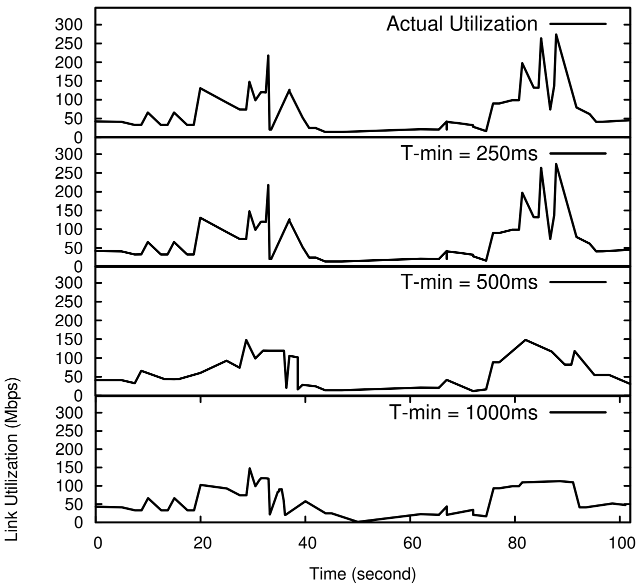

Fig. 10. Effect of T_{min} on Measured Utilization

3) Effect of Minimum Polling Frequency, $\mathcal{T}_{min}$: As explained in Algorithm 1, our scheduling algorithm adopts to the traffic pattern. For, a rapidly changing traffic spike, the poling frequency sharply decreases and reaches $\mathcal{T}_{min}$. In Fig. 10, we present the impact of $\mathcal{T}_{min}$ on monitoring accuracy. Evidently, the monitoring data is very accurate for $\mathcal{T}_{min} = 250ms$ and it gradually degrades with higher values of $\mathcal{T}_{min}$. However, monitoring accuracy comes at the cost of network overhead as presented in Fig. 11. This figure presents the root-mean-square (RMS) error in monitoring accuracy along side the messaging overhead for different values of $\mathcal{T}_{min}$. This parameter can be adjusted to trade-off accuracy with messaging overhead, depending on the application requirements.

VII. Conclusion and Future Work

In this paper, we have introduced PayLess – a flexible and extendable monitoring framework for SDN. To the best of our knowledge, PayLess is the only monitoring framework for SDN. Almost every aspect of monitoring can be specified using PayLess's generic RESTful API. Moreover, the core components in PayLess framework can be replaced by custom implementations without affecting the other components. To demonstrate the effectiveness of PayLess framework, we have presented an adaptive scheduling algorithm for flow statistics collection. We implemented a concrete use case of monitoring link utilization using the proposed algorithm. We have evaluated and compared its performance with that of Flowsense and a periodic polling method. We found that the proposed algorithm can achieve higher accuracy of statistics collection than FlowSence. Yet, the incurred messaging overhead is upto 50% of the overhead in an equivalent periodic poling strategy. Our long term goal along this work is to provide an open-source, community driven monitoring framework for SDN. This should provide a full-fledged abstraction layer on top of the SDN control platform for seamless network monitoring application development. We also plan to demonstrate the effectiveness of our platform by implement an autonomic QoS policy enforcement application [26] on top of PayLess and perform large scale experiments in our OpenFlow testbed [27]. It is also one of our future goals to make PayLess compatible with distributed controller platforms [28].

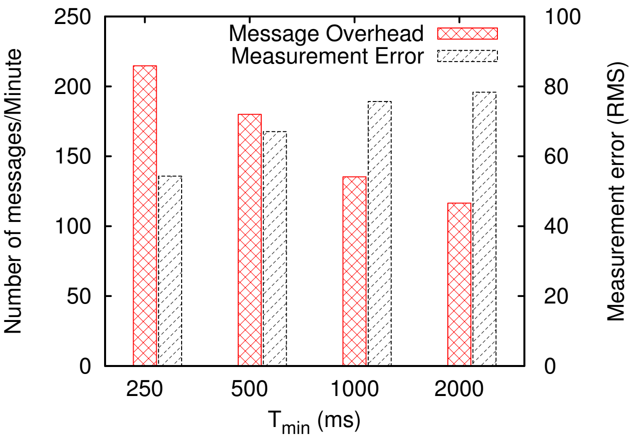

Fig. 11. Overhead and measurement error

Acknowledgement

This work was supported by the Natural Science and Engineering Council of Canada (NSERC) in part under its Discovery program and in part under the Smart Applications on Virtual Infrastructure (SAVI) Research Network.

References

[1] "Cisco NetFlow site reference," http://www.cisco.com/en/US/products/ps6601/products_white_paper0900aecd80406232.shtml.

[2] "Traffic Monitoring using sFlow," http://www.sflow.org/.

[3] A. C. Myers, "JFlow: Practical mostly-static information flow control," in *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1999, pp. 228–241.

[4] C. Systems, "Cisco CNS NetFlow Collection Engine," http://www.cisco.com/en/US/products/sw/netmgtsw/ps1964/index.html.

[5] H. Kim and N. Feamster, "Improving network management with software defined networking," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 114–119, 2013.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[7] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring Network Utilization with Zero Measurement Cost," in *Passive and Active Measurement*. Springer, 2013, pp. 31–41.

[8] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proceedings 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI*, vol. 13, 2013.

[9] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: traffic matrix estimator for OpenFlow networks," in *Passive and Active Measurement*. Springer, 2010, pp. 201–210.

[10] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches," in *Proc. of the USENIX HotICE workshop*, 2011.

[11] Y. Zhang, "An adaptive flow counting method for anomaly detection in sdn," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 25–30.

[12] M. Moshref, M. Yu, and R. Govindan, "Resource/Accuracy Tradeoffs in Software-Defined Measurement," in *Proceedings of HotSDN 2013*, August 2013, to appear.

[13] A. Jain and E. Y. Chang, "Adaptive sampling for sensor networks," in *Proceeedings of the 1st international workshop on Data management for sensor networks: in conjunction with VLDB 2004*. ACM, 2004, pp. 10–16.

[14] B. Gedik, L. Liu, and P. Yu, "Asap: An adaptive sampling approach to data collection in sensor networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 12, pp. 1766–1783, 2007.

[15] A. D. Marbini and L. E. Sacks, "Adaptive sampling mechanisms in sensor networks," in *London Communications Symposium*, 2003.

[16] J. Kho, A. Rogers, and N. R. Jennings, "Decentralized control of adaptive sampling in wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 3, p. 19, 2009.

[17] C. Alippi, G. Anastasi, M. Di Francesco, and M. Roveri, "An adaptive sampling algorithm for effective energy management in wireless sensor networks with energy-hungry sensors," *Instrumentation and Measurement, IEEE Transactions on*, vol. 59, no. 2, pp. 335–344, 2010.

[18] R. Willett, A. Martin, and R. Nowak, "Backcasting: adaptive sampling for sensor networks," in *Information Processing in Sensor Networks, 2004. IPSN 2004. Third International Symposium on*, 2004, pp. 124–133.

[19] E. Hernandez, M. Chidester, and A. George, "Adaptive sampling for network management," *Journal of Network and Systems Management*, vol. 9, no. 4, pp. 409–434, 2001.

[20] G. Androulidakis, V. Chatzigiannakis, and S. Papavassiliou, "Network anomaly detection and classification via opportunistic sampling," *Network, IEEE*, vol. 23, no. 1, pp. 6–12, 2009.

[21] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110.

[22] "POX OpenFlow Controller," https://github.com/noxrepo/pox.

[23] "Floodlight openflow controller," http://www.projectfloodlight.org/floodlight/.

[24] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 1–12.

[25] "Iperf: TCP/UDP Bandwidth Measurement Tool," http://iperf.fr/.

[26] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*. IEEE, 2013, pp. 1–7.

[27] A. R. Roy, M. F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba, "Design and Management of DOT: A Distributed OpenFlow Testbed," in *14th IEEE/IFIP Network Operations and Management Symposium (NOMS 2014)*.

[28] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic Controller Provisioning in Software Defined Networks," in *9th International Conference on Network and Service Management 2013 (CNSM 2013)*, Oct 2013, pp. 18–25.