IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 12 July 2023, accepted 2 August 2023, date of publication 9 August 2023, date of current version 15 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3303847

RESEARCH ARTICLE

Graph Modeling for OpenFlow Switch Monitoring

ALI MALIK AND RUAIRÍ DE FRÉIN
School of Electrical and Electronic Engineering, Technological University Dublin, Dublin 7, D07 H6K8 Ireland
Corresponding author: Ali Malik (ali.malik@tudublin.ie)

This work was supported by the Science Foundation Ireland (SFI) under Grant 15/SIRG/3459 and Grant 13/RC/2077_P2.

ABSTRACT Network monitoring allows network administrators to facilitate network activities and to resolve issues in a timely fashion. Monitoring techniques in software-defined networks are either (i) active, where probing packets are sent periodically, or (ii) passive, where traffic statistics are collected from the network forwarding elements. The centralized nature of software-defined networking implies the implementation of monitoring techniques imposes additional overhead on the network controller. We propose Graph Modeling for OpenFlow Switch Monitoring (GMSM), which is a lightweight monitoring technique. GMSM constructs a flow-graph overview using two types of asynchronous OpenFlow messages: `packet-in` and `flow-removed`, which improve monitoring and decision making. It classifies new flows based on the class of service. Experimental findings suggest that using GMSM leads to a decrease in network overhead resulting from the communication between the controller and the switches, with a reduction of 5.7% and 6.7% compared to state-of-the-art approaches. GMSM reduces the controller's CPU utilization by more than 2% compared to other monitoring methods. Overhead reduction comes with a slight reduction of approximately 0.17 units in the estimation accuracy of links utilization because GMSM allows the user to monitor the network subject to a selected class of service, as opposed to having an exact view of the network utilization.

INDEX TERMS Software-defined networking, OpenFlow, monitoring, overhead, utilization.

I. INTRODUCTION

Network monitoring is a fundamental requirement for many networking systems. To gain a deeper insight into the network's traffic, a traffic monitoring system can engage in various activities including but not limited to traffic matrix estimation, recognising elephant flows and determining link utilization. These activities produce reports which are gathered using a real-time monitoring application, for example [1], which is designed for network appliances, and can be scheduled to run at fixed or dynamic periodic intervals. Monitoring is beneficial for a wide range of network applications, including but not limited to security, traffic engineering and load balancing [2].

Measurement is a separate task from monitoring [3]; the focus of this paper is monitoring. The interaction between monitoring and measurement functions is described using the following example. Transmitting data from delay or jitter sensitive applications gives rise to the need for path latency quality measurements to ensure that service level requirements are met. A classical example of monitoring in this case is the Real-Time Transport Control Protocol (RTCP), which provides feedback in the form of reports, for the jitter sensitive Real-time Transport Protocol (RTP) [4], helping to ensure that maximum latency thresholds for interactive sessions are met.

The delivery of monitoring messages induces an overhead which can be expressed as the cost associated with delivering information about network state [3] to the monitoring function. In this paper, we define the overhead as the number of packets that are required to collect the statistics from forwarding elements. Traditionally, serviceability and bespoke operations are the main features of monitoring approaches [5], [6], [7]. However, the decentralized nature of traditional networking systems usually leads to a partial view of the network. For example, the RTCP-based feedback described above provides information about a service delivered on one path across the network. It does not provide information about other services which are multiplexed over the same resources. Emerging networking technologies such

The associate editor coordinating the review of this manuscript and approving it for publication was Jose Saldana.

as cloud computing and vehicular networks require efficient monitoring due to the increased dynamicity of the demands placed on these networks [8]. The view provided should also facilitate an analysis of the interaction between the different processes which are multiplexed over the network.

Software-Defined Networking (SDN) is a promising networking architecture for centralized management. SDN-based monitoring has the potential to be flexible due to the elastic nature of the SDN architecture compared to the rigidity of traditional networks. In centrally controlled networks like SDNs, network flows can be monitored using an `active` approach, which is often also called a polling approach. In this method, the controller regularly issues polling messages to a predefined set of forwarding elements to collect flow statistics. Alternatively, a `passive` approach which is also known as switch pushing is used. In this case, flow statistics are received by the controller from the node(s) whose flow table encountered flow completions [9]. This can be achieved by utilizing the Application Programming Interface (API) of a southbound protocol such as the standard OpenFlow [10] that defines the communication between the controller and data plane elements. Both active and passive monitoring have drawbacks that need to be considered.

Polling is a widely-used monitoring approach that involves the SDN controller directing switches in its domain to report their statistics using a set of controller-to-switch messages. Using these messages, the controller can send multiple queries to the switches. These messages have different objectives, for instance the `table_features`, `port_stats` and `flow_monitor` messages enable the controller to obtain information about the status of flow tables, ports and flow changes respectively. In response to these queries, switches usually respond to the controller with reports on the current state of the recorded statistics. The drawback of regular querying of network forwarding elements is that it causes a significant increase in controller resource usage, which is quantified using an overhead score. If the set of network switches is $V = \{v_1, v_2, \ldots v_n, \ldots\}$, where the total number of switches is denoted by $|V|$, the number of polling messages, $p[n]$, that an SDN controller needs to handle at the n^{th} monitoring period is expressed as

$$p[n] = \sum_{n=1}^{|V|} s[n] + \sum_{n=1}^{|V|} r[n], \tag{1}$$

where $s[n]$ and $r[n]$ denote the sent and received polling messages respectively for the set of network switches V during the n^{th} monitoring period. Monitoring can be performed on a subset of switches that hold active flows or based on another predefined monitoring condition.

The switch pushing approach depends on flow arrivals and completions [11]. This approach only depends on the switch-controller `packet_in` and `flow_removed` messages. When a new packet arrives, the switch triggers a `packet_in` message that notifies the controller to install a new forwarding rule that matches the new flow and a new path is established. Usually, the installed rules have an associated timeout flag, which indicates its expiration time. Rule expiration can be `soft`, in which the rule is removed when it becomes idle, or `hard`, in which the rule is removed after a predefined timeout. When the timeout of an installed flow entry expires, the `flow_removed` message is triggered. Flow arrival and completion messages are used to gather information about the size and duration of flows. They are matched against entries which can be obtained from the network forwarding elements. The disadvantage of switch pushing is that it may not provide instantaneous reports about existing flows. This is because monitoring data gathered this way does not capture the true state when flows are in operation. When the frequency of gathering monitoring reports from all network appliances is high, good information about the network state can be acquired and an informative network analysis can be performed. However, this will incur a high overhead on the network which has negative impacts on resources such as the controller's CPU utilization.

In this paper, a new SDN monitoring system, called GMSM, is introduced. The prototype system provides a visual presentation that integrates the information collected via monitoring, which helps the user to decide which data plane elements should be included in future monitoring. The object of the GMSM approach is to reduce controller overhead while maintaining a high level of information about the state of data plane appliances.

GMSM differs from the state-of-the-art methods for its capability of running a non-blind monitoring. This is done by classifying the class of service for the incoming flows and to be modelled as a graph to provide an additional privilege for the network managers.

This paper is organised as follows. In Section II SDN monitoring techniques are presented and discussed. In Section III, we introduce GMSM. Performance evaluation results are presented in Section IV. Suggestions for future work directions are made in the context of the main results of this paper, in Section VI.

II. RELATED WORK

NetFlow [5], sFlow [6] and JFlow [7] are the leading flow based network monitoring solutions in traditional IP networks. Both NetFlow and JFlow are proprietary solutions and licensing costs are a barrier to their uptake. sFlow is not widely deployed.

Given the recent surge of interest in SDN and QoS-routing technologies, network monitoring has received considerable attention. The authors in [12] presented OpenTM as an SDN traffic matrix estimator. OpenTM gathers the traffic statistics for each active flow by sending status request messages at regular intervals. These active flows are detected according to the routing information available in the network controller.

In their work on a low-cost monitoring technique for SDNs, the authors of [13] proposed CeMon. CeMon polls the statistics from a set of target switches. This set is established by determining the active flows in the network.

TABLE 1. Flow statistics reporting methods are categorized as: PP (Periodic-Polling), AP (Adaptive-Periodic), SP (Switch-Pushing) and H (Hybrid).

Technique	Visualization	Traffic Differentiation	Reporting	Main Purpose
OpenTM [12]	✗	✗	PP	Selective querying methods for estimating traffic matrix.
CeMon [13]	✗	✗	AP	Monitors link utilization and estimates traffic with low overhead.
OpenLh [14]	✗	✗	AP	Monitors active flows to reduce the overhead.
FlowSense [11]	✗	✗	SP	Monitors the links utilization with no overhead cost.
PayLess [15]	✗	✗	PP	Monitors links utilization and reduces the messaging overhead.
EPS [16]	✗	✗	H	High accuracy in real-time statistic monitoring with low overhead.
CO-FSC [17]	✗	✗	PP	Reducse monitoring bandwidth consumption and processing delay.
IPro [18]	✗	✗	AP	Reduces the overhead and the CPU usage.
APAM [19]	✗	✗	AP	Minimizes overhead based on switch's port utilization.
COCO [20]	✗	✗	AP	Reduces the monitoring message count. Yields high accuracy.
MWCC [21]	✗	✗	PP	Reduces overhead and delivers accurate flow statistics collection.
LCM [22]	✗	✗	PP	Minimizes the cost of both flow monitoring and reporting delay.
Opimon [23]	✓	✗	PP	Monitors flow tables and switch statistics.
MVT [24]	✓	✗	PP	Interactive monitoring for network consumption.
OF@TEIN [25]	✓	✗	PP	Monitors and plots metrics using sFlow-RT.
GMSM	✓	✓	H	Reduces overhead by prioritizing the monitored flows.

The main motivation is to reduce the communication cost by generating a cost effective polling scheme. In their work on an OpenFlow-based low-overhead and high-accuracy framework, the authors of [14] proposed OpenLh. OpenLh reduces the network overhead by adopting an adaptive periodic polling algorithm that works based on the change rate of the future flows.

FlowSense is a link utilization monitoring solution that incurs low overhead [11]. It uses two key OpenFlow messages, `packet-in` and `flow-removed`, which are automatically generated by switches upon the arrival or termination of flows. The authors claim that it imposes no overhead on the SDN controller because it does not use a periodic polling approach. PayLess, an SDN query-based real-time monitoring framework which was introduced in [15], adopts an adaptive scheduling polling approach that reduces the network controller overhead. It provides a RESTful API that enables other SDN applications to take advantage of monitoring data.

In [16], the authors presented the Elastic Polling Scheme (EPS), which functions as a real-time polling system for software-defined data center networks. The goal of EPS is to monitor the network accurately and efficiently, with minimal controller involvement. The experimental results showed that EPS is capable of reducing overhead more effectively than CeMon and PayLess. Another cost-optimized flow statistics collection process for SDN called CO-FSC was introduced in [17]. CO-FSC relies on wild card requests that are sent by the controller to collect partial flow statistics from a set of target switches. Only flows that match the wild card rule are collected, improving monitoring by reducing both the utilized bandwidth and processing delay. IPro, an intelligent probing system for SDN monitoring, was introduced in [18], which tunes the probing intervals of its monitoring approach using a reinforcement learning algorithm so that the monitoring overhead is reduced. APAM is a lightweight active-port aware approach to monitoring introduced in [19], which identifies the active ports first and then adjusts the polling interval based on the utilization of each active port.

In their paper [20], the authors introduced a new SDN adaptive monitoring technique called the information COnfidence index COllection (COCO). COCO aims to reduce the communication overhead while collecting statistics. Their experimental results showed that with the proposed method the exchanged polling messages can be reduced by 55-70% compared to the periodic polling. The authors of [21] proposed MWCC, which is a multi-path weighted closeness centrality-based approach for flow statistics collection. MWCC reduces the monitoring overhead by ranking the network forwarding elements based on the state of the underlay topology. The top k-ranked elements are selected in the statistics collection process. Preliminary work, in [22], on formulating the problem of selecting the nodes involved in the statistics collection process, consisted of expressing the problem as an integer linear program. Low Cost Monitoring (LCM) was proposed as a heuristic solution to address the problem of minimizing both the monitoring cost and the reporting delay.

Several studies have used SDN visualization as a part of monitoring. The authors in [23] proposed Opimom, which is a transparent monitoring system for SDNs that imposes some overhead on performance. Opimon is controller API independent, which makes it compatible with any OpenFlow network. A web interface presents a real-time network visualization. The authors in [24] presented an approach for monitoring the resource consumption periodically. This was achieved by monitoring the idle and active forwarding rules. They visualized the collected information and made it available for the network administrator via a Model-View Template (MVY). The authors of [25] presented OpenFlow@Trans Eurasia Information Network (OF@TEIN). By utilizing sFlow-RT, a wide network visibility was provided.

In Table 1, we summarize these solutions, highlighting their objectives under the headings: link utilization, overhead reduction, bandwidth minimization and traffic matrix estimation. Table 1 shows the incorporated visualization as well as the nature of the reporting frequency.

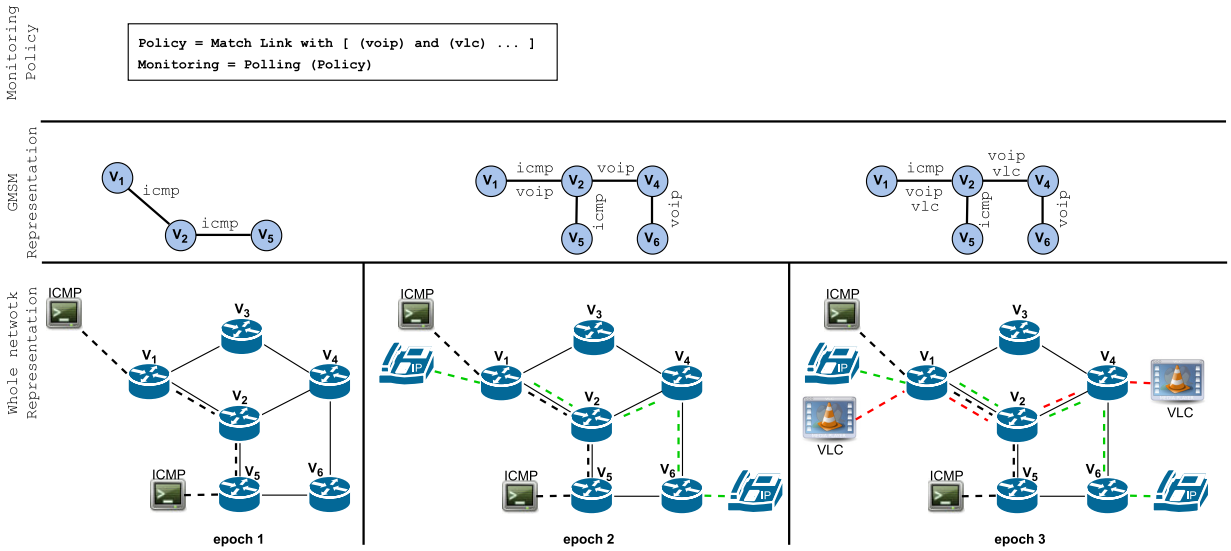

FIGURE 1. Motivating example for GMSM: The underlying network's components, routers and hosts, are illustrated in the lowest layer. During the three epochs illustrated, different types of traffic are delivered, ICMP (black dashed lines), VoIP (green dashed lines), and finally, video (red dashed lines). GMSM constructs a graph of the active traffic flows using `packet-in` and `flow-removed` OpenFlow messages. This flow graph abstraction that represents the ongoing traffic flows and the CoS is illustrated in the middle layer for the three epochs. The monitoring policy that determines how the flow graph is constructed is illustrated in the top layer.

Research interest in SDN monitoring is growing. Many of the contributions so far have focused on reducing the overhead. In this paper, we aim to reduce the overhead of statistics monitoring, whilst also providing an abstraction layer from which monitoring decisions regarding remedial actions can be made. These contributions are summarized as follows.

- We propose a new network model that allows the SDN controller to classify the incoming traffic and to provide filtered information to GMSM.
- Our aim in designing GMSM is to lower the network overhead which is caused by monitoring while ensuring that the monitoring process is not performed blindly. Therefore, we perform real-time statistics monitoring in a manner which is correlated with the traffic types observed.
- We provide an open-source implementation of the GMSM framework and algorithm. We test the efficiency of GMSM and provide evidence that GMSM improves SDN-based monitoring.

III. PROPOSED METHOD: GMSM

Previous SDN monitoring studies attempted to reduce overhead by optimizing the polling intervals used for monitoring. Polling was conducted blindly. The Class of Services (CoS) and the current graph of the traffic flows was not known. We construct a flow graph abstraction to represent the ongoing traffic flow and the CoS being delivered using `packet-in` and `flow-removed` OpenFlow messages as follows. Elements can be added to the flow graph when a new flow arrives. When a matching forwarding rule does not exist an OpenFlow `packet-in` message is sent to the controller requesting a path setup. Elements can be removed from the flow graph when a traffic flow terminates and the forwarding rule expires. The concerned switches generate `flow-removed` messages informing the controller about the flow table updates. It is not necessary for there to be a one-to-one mapping between the constructed graph and the underlay network topology. Constructing the flow graph with information about the CoS of applications running on the network is of benefit. The first k packets of each flow are classified as they arrive, to identify its CoS. The CoS of each flow is mapped in the flow graph so that an overview of the traffic type that is delivered over each link in the network is available.

Fig. 1 depicts a network with 6 nodes and 7 links, which serves as a toy example that illustrates the construction of a flow graph in GMSM. The figure is divided into three layers. The bottom layer represents the physical data plane appliances along with the existing flows being transmitted over the network. This layer includes all the network elements, constructed based on the global view of the SDN controller. The middle layer represents the flow graph that GMSM constructs based on existing active flows, classified by their Class of Service (CoS). In this layer, the inactive elements are eliminated from the network graph. The top layer represents the management of monitoring information, which is defined as a network policy in which the network administrator decides the events of interest to be captured during the monitoring polling process. In this layer, single or combined CoS events can be created so that the polling will be more focused on the predefined monitoring policy rather than considering all events.

We illustrate three different epochs, during which the network supports different CoS applications. During epoch 1, only one Internet Control Message Protocol (ICMP) flow

passes through the network via the path (v_1, v_2, v_5). During epoch 2, the network starts carrying a new Voice over Internet Protocol (VoIP) flow over the path (v_1, v_2, v_4, v_6). Finally, during epoch 3, a new VideoLAN Client (VLC) streaming flow starts. It is delivered over the path (v_1, v_2, v_4). Fig. 1 also shows the status of the flow graph at each epoch, including nodes and links with CoS details. To reduce the overhead associated with collecting flow statistics, the information in the flow graph is used during monitoring. For example, the flow graph can be used as an estimator for outage hot spot areas, where particular attention is given to critical services that require high QoS.

A. NETWORK MODEL

The network is modeled as an undirected graph, G, [26]. The graph, G, is composed of a set of vertices and a set of edges, $G = (V, E)$. The vertex-set, V, represents the finite set of switches, $\{v_1, \ldots, v_n\}$ in the network, where v_i is the i-th switch. The edge-set, E, represents the finite set of bidirectional links that connects the switches. The set of all edges in the graph is a subset of the set of all possible pairs of vertices, $E \subseteq V \times V$. A path P_{ij} is a route between the two switches, v_i and v_j. We call v_i and v_j the source and destination switches of the path, P_{ij}. The path P_{ij} is defined as the sequence of vertices traversed in order to get from v_i to v_j. For example, the path (v_i, v_p, v_q, v_j) consists of four vertices and three edges, $\{v_i, v_p\}$, $\{v_p, v_q\}$ and $\{v_q, v_j\}$. Pairs of vertices in a path are members of the edge-set, $\{v_i, v_{i+1}\} \in E, \forall 1 \leqslant i < j$. The path, P_{ij}, is a simple path if all of its switches are distinct. We assume that flows traverse simple paths. The cost associated with traversing the path, P_{ij}, is the sum of the cost of its constituent edges, that is $C(P_{ij}) = \sum_{\{v_i, v_{i+1}\} \in P_{ij}} C(\{v_i, v_{i+1}\})$. The edge cost can be any additive metric such as the hop count, delay or usage. The shortest path from v_i to v_j is the path with the smallest cost out of all possible paths from v_i to v_j. The set of traffic classes is $CoS = \{c_1, c_2, \ldots, c_n\}$, for example, VoIP, ICMP or Video. Each edge in a path is mapped to one or more CoS items according to an one-to-many function $f : E \mapsto CoS$.

B. GMSM FRAMEWORK AND ALGORITHM

The architecture for GMSM is illustrated in Fig. 2. The GMSM framework is implemented as an SDN northbound application. Pseudo-code for GMSM is listed in Alg. 1. It details how low overhead, integrated-view monitoring is achieved. In addition to the management layer that allows the network operator to define the monitoring policies and to configure the network, the architecture is composed of three main components, the Network Controller, a Topology Discovery component and the GMSM component, which carry out the following functions:

1) Network Controller: The controller is the network brain where decision making and intelligence is located. We use the POX controller as it facilitates fast prototyping [27]. The standard OpenFlow protocol is used as a southbound API for establishing the communication between the data and control planes, whereas the set of POX APIs is used on the northbound interface to develop network control applications.

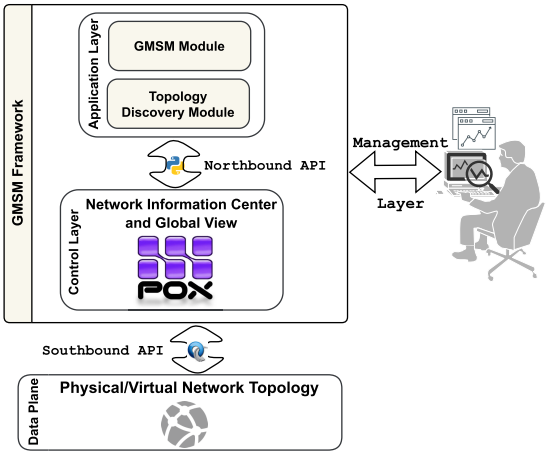

FIGURE 2. Architecture of the proposed framework and its components: the primary contribution of this paper is the GMSM block. Openflow is used on the southbound interface and POX APIs are used on the north bound interface.

Algorithm 1 GMSM

▷ **Input :** Network topology, $G(V, E)$. Initialize the empty flow graph, $G_1(V, E) = \emptyset$

▷ **Constructing Flow Graph** G_1 **:**

foreach `packet_in` **do**
  $P_{ij} = Dijkstra(v_i, v_j) \parallel$ capture k packets from v_i
  Classify the CoS of the k packets
  for *each* $\{v_i, v_{i+1}\} \in P_{ij}$ **do**
    if $\nexists \{v_i, v_{i+1}\} \in G_1$ **then**
      $G_1 \leftarrow [\{v_i, v_{i+1}\},\ data = CoS]$
    else
      $\{v_i, v_{i+1}\} \leftarrow\!\!\shortmid\ data = CoS$

foreach `flow_removed` **do**
  Detect terminated flows $\bar{P}_{ij}$
  for *each* $\{v_i, v_{i+1}\} \in \bar{P}_{ij}$ **do**
    $data = data - CoS$
    if $data(v_i, v_{i+1}) = \emptyset$ **then**
      $G_1 = G_1 - \{v_i, v_{i+1}\}$

▷ **Monitoring :**

foreach $[(v_i, v_{i+1}) \in G_1] \wedge [(v_i, v_{i+1}) \geqslant Threshold]$ **do**
  Send `stats_request` every n seconds

2) Topology Discovery Module: This component is responsible for discovering the underlying network topology information so that such information will be available for other applications. The standard POX discovery module is used [28]. We use NetworkX [29] to represent and manipulate the network topology as a graph.

3) GMSM Module: The GMSM protocol is used to construct a flow graph that: (i) determines the CoS on each link in G; and (ii) polls the statistics from a certain set of

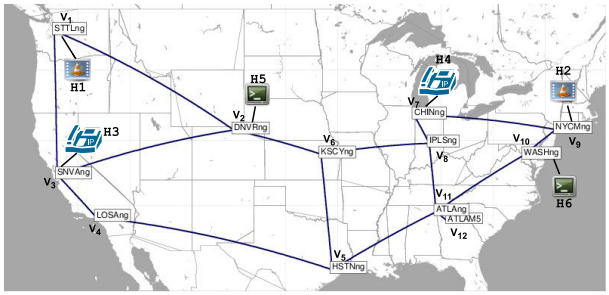

FIGURE 3. Monitoring algorithms are evaluated using the Abilene topology with $|V| = 12$ switches, $v_1, \ldots v_{12}$, $|E| = 15$ edges and hosts $H_1, \ldots H_6$.

switches that are defined by the monitoring policy of the network's administrator.

We continue by describing the pseudo-code for GMSM in Alg. 1. GMSM consists of three phases of operation, a `packet_in` driven phase, a `flow_removed` phase and finally, a polling phase. The worst case computational complexity of Dijkstra's algorithm bounds the computational complexity of Alg. 1. Using a Fibonnaci heap the worst case computational complexity for the entire graph G is $O(|E| + |V|log(|V|))$.

Lines 2-12 describe the initial switch pushing phase of GMSM. GMSM constructs the flow graph, which is denoted G_1, using incoming `packet_in` requests. The shortest path between the source and destination of new flows is determined using Dijkstra's algorithm. GMSM gathers k packets from each new flow for the purpose of classification. We use $k = 2$ packets in this classification step. Terminated flows are denoted $\bar{P}_{ij}$.

In lines 13-21, the flow graph, G_1, is updated by removing flows that have been terminated, $\bar{P}_{ij}$, along with each terminated flow's associated CoS information. The operation, $G_1 = G_1 - \{v_i, v_{i+1}\}$, indicates how the link $\{v_i, v_{i+1}\}$, which is a component of the terminated flow, $\bar{P}_{ij}$, is removed from the flow graph G_1.

The polling phase is described in lines 22-24. In the polling phase GMSM sends requests to switches whose links are carrying a CoS that meets or exceeds the threshold. The threshold is a service specific value which is set by the manager. Examples for a video delivery application include the count of active flows, delay or jitter threshold. The threshold is compared with a measurement of the service on the link, $\{v_i, v_{i+1}\}$, which is denoted $m(v_i, v_{i+1})$. When $m(v_i, v_{i+1})$ quantifies the number of active RTP flows, it returns an integer value which is compared with the threshold. This process is described in lines 22-24. The threshold is set by the network administrator using the system prototype. In this paper, we selected the Real-time Transport Protocol (RTP) as the CoS. The threshold was tuned empirically by evaluating the number of active RTP flows on a link.

Graph modelling in GMSM incurs no additional overhead. This is because GMSM relies on the procedure of path computation to update the graph when there is a new arrival request. It relies on G_1, which is constructed and updated in response to two event-types: (i) a new arrival request; and (ii) traffic termination. It relies on switch pushing when a certain types of traffic in the CoS terminate. GMSM's graph modeling provides an important, interactive overview layer for existing services that are of interest to the network operators. GMSM makes switch querying `non-blind` in the network monitoring process. The implementation code for GMSM is available on GitHub (`https://github.com/Ali00/GMSM`).

TABLE 2. Experimental Specifications.

Specification	Details
Emulator	Mininet V. 2.2.2
OpenFLow protocol	V. 1.0
POX	V. 0.2.0 carp
D-ITG	V. 2.8.1 Revision: 1023
VLC traffic	Big-Buck-Bunny.mp4
SIPp	V. 3.4
Configuration	Out-of-Band
Topology	Abilene US
Polling interval	1s

IV. EXPERIMENTAL SETUP AND DESIGN

Our hypothesis is that GMSM, which is a hybrid active flow polling algorithm, outperforms active flow and polling monitoring techniques in terms of the network overhead. We compare GMSM with two benchmark approaches: an implementation of the OpenTM [12] approach and a periodic fixed-polling approach in which all switches are queried. We investigate the sensitivity of flows to different background traffic types. We then evaluate GMSM using two metrics: (1) the controller's overhead and (2) the estimated links utilization.

We used a real-world network topology called Abilene from the Survivable fixed telecommunication Network Design library (SNDlib) [30], which is illustrated in Fig. 3. The link capacity was set to 8 Mbps. Six clients were added to the original topology, $H_1, \ldots H_6$, to inject and receive traffic. Three types of CoS were used to evaluate our hypothesis. The CoS used were VLC streaming, VoIP and ICMP. To stream video we used VLC servers to stream the "Big Buck Bunny" MPEG-4 video, which is approximately 10 min. long, using the RTP [4] between H_1 and H_2. The H.264 video compression standard for high-definition digital video was used. The open-source SIPp [31], which is a performance testing tool for the Session Initiation Protocol (SIP), was utilized to generate Internet telephony calls VoIP between H_3 and H_4. The Distributed Internet Traffic Generator (D-ITG) [32] was used to generate ICMP traffic, a non-RTP type of flow, in the network between the clients H_5 and H_6. The inter-departure time of ICMP was set to a constant rate of 1000 packets per second with a packet size of 512 B. We emulated the network using Mininet [33]. Mininet is a widely used emulation system for emulating SDN architectures in various experimental scenarios as well as to evaluate and prototype new protocols and applications [34].

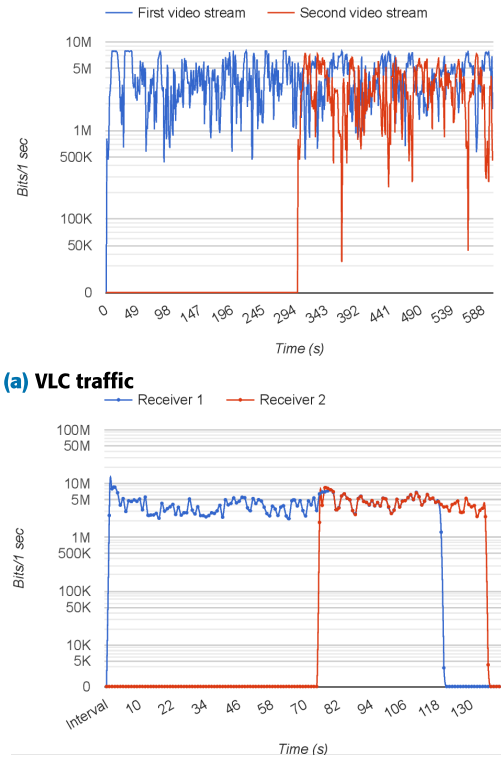

FIGURE 4. Evaluation of two different CoS flows: video and ICMP. Information is captured from the $\{v_2, v_6\}$ link of the Abilene topology. The video and ICMP flow traverse a common path $(v_1, v_2, v_6, v_8, v_7, v_9)$. This figure shows that one CoS, video, is significantly affected by the addition of a second flow compared to ICMP.

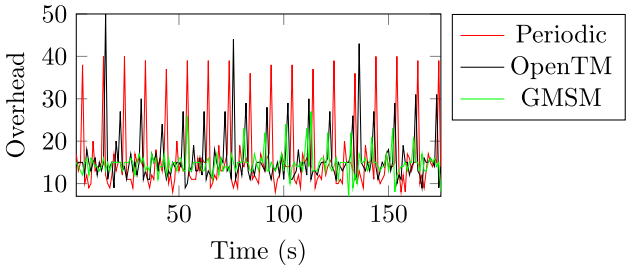

FIGURE 5. Quantifying overhead: the total number of invoked OpenFlow messages are 2578, 2735 and 2764 messages for GMSM, OpenTM and periodic monitoring respectively.

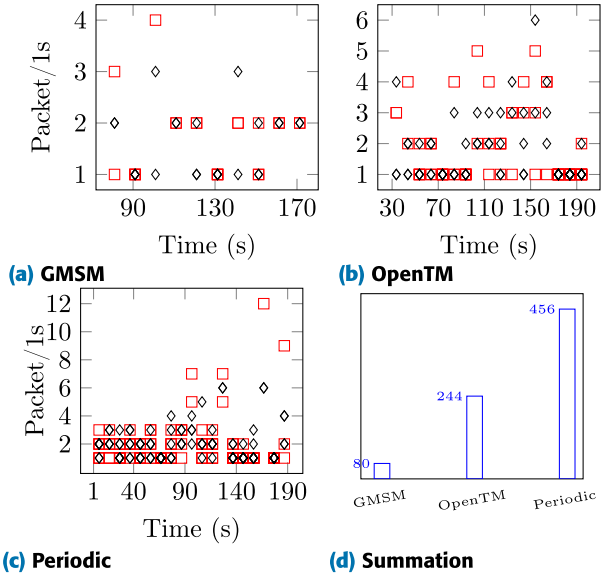

FIGURE 6. The count of OpenFlow stats request/reply packets per 1s is illustrated as a function of time. In this experiment the monitoring function gathers monitoring reports at 10 s intervals. The symbol, □, represents the `stats_request` and the symbol, ◇, represents `stats_reply`. GMSM reduces the overhead by reducing the number of switches that participate in the monitoring process.

In the emulation environment, we employed two servers; one acted as the OpenFlow controller and the other simulated the network topology. For each server, we used Ubuntu v.14.04 LTS with Intel Core-i5 CPU and 8GB RAM. Table 2 specifies the parameters used in the experiments.

A. PRELIMINARY EVALUATION

Since RTP flows are more sensitive to networking incidents such as delay or jitter, we focused on this class of traffic as a first scenario and determined a suitable GMSM monitoring threshold.

Two live VLC streams were launched over one single common path of 8 Mbps link capacity. The experiment lasted for 10 min. The first video started at time 0 min. and ended at time 10 min. The second video started at time 5 min. and ended at time 10 min. Fig. 4(a) shows the bit rate of both videos. The bit rate of the second video is affected by the presence of the first video stream, which is observed by its lower bit rate. This results in a lower quality of delivery for the second video. On average, the bit rate of first video is 0.5 Mbps for the first 5 min. and 0.49 Mbps for the rest of the experiment. The average bit rate of the second video is 0.42 Mbps. This 8% difference affected the video quality.

Another experiment was conducted to see how ICMP traffic interfered with other ICMP traffic, to complement the VLC scenario described above. The ICMP experiment ran for approximately 2 min. The first ICMP flow, which is labelled Receiver 1 in Fig. 4(b), ran for 2 min. The second ICMP, which is labelled Receiver 2, started after 1 min and ran for approximately 1 min. Fig. 4(b) shows the bit rate of both flows. On average, the bit rate of the two ICMP flows are approximately equal, 0.48 Mbps. Compared to the video bit rate, the ICMP bit rate was not significantly affected by the other flow. We conclude that RTP flows require a greater priority than ICMP flows as their quality of delivery is significantly affected when the bit rate is reduced [35]. Consequently, we set the GMSM threshold to be 2 or more RTP flows on a link.

To estimate the overhead and network utilization for the three monitoring schemes, we ran additional experiments for approximately 3 min. At time 0 min, only the video stream had started; at time 1 min, the VoIP flow started and at time 2 min, the ICMP flow had started. All flows terminated at the same time. This mix of traffic types allows us to investigate monitoring an application such as video which has strict QoS requirements in the presence of traffic types which do not

have the same requirements. GMSM's application-oriented threshold makes it particularly suitable for this.

V. EXPERIMENTAL RESULTS

This section presents the simulation results that evaluate the performance of the proposed GMSM framework. The evaluation aims to determine two key performance indicators. Firstly, we quantified the network overhead with respect to the volume of query messages and its impact on CPU utilization. Secondly, we evaluated the impact of the collected statistics on network analysis efficiency.

A. QUANTIFYING NETWORK OVERHEAD

Since we defined the overhead as the amount of openflow messages that generated to gather various statistics from the OpenFlow switches, this means that to assess the network overhead, we first quantified the volume of OpenFlow messages that were generated due to monitoring events. Fig. 5 shows all OpenFlow generated messages during one experiment. We examined the OpenFlow messages by type, `stats_request` and `stats_reply`, in Fig. 6. GMSM outperformed both periodic polling and OpenTM methods by reducing the number of the generated OpenFlow messages.

The generated OpenFlow messages also served as an indicator of the usage of controller resources such as the CPU. To determine the controller's CPU utilization, the standard linux tool, `vmstat`, was employed to observe the POX controller machine. It collected and logged summary information about resource usage such as CPU and memory. `vmstat` reported the CPU idle time percentage in 1 second intervals. The CPU utilization was calculated by subtracting the percentage of time the CPU was idle from 100%. POX's CPU utilization was assessed at 1 second intervals during the experiments. Fig. 7 shows the controller CPU utilization for the three monitoring schemes. On average, the periodic scheme accounted for 21.6% of CPU usage, while the OpenTM and GMSM schemes accounted for 21.2% and 18.9% of the CPU respectively. It can be clearly seen that GMSM can lower the network overhead by the means of (i) the OpenFlow messages and (ii) the usage of controller's CPU.

B. NETWORK ANALYSIS CAPACITY

The monitoring schemes used to report network performance can vary, and as a result, the knowledge gained by the SDN controller will differ depending on the information provided by each scheme. Having access to a greater amount of information allows for a deeper understanding of the network status, thus enabling more precise network analysis. We use the bandwidth utilization as a metric to evaluate the effectiveness of each monitoring scheme in providing accurate network analysis. To accomplish this we compare the accuracy of the network utilization estimated using GMSM with the periodic polling and OpenTM methods. At each timestamp, we look at the network utilization of all links whose end switches participated in the monitoring

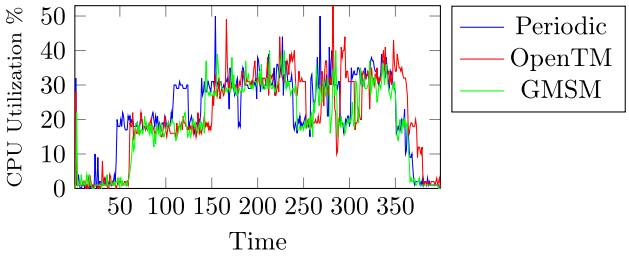

FIGURE 7. POX's CPU utilization when GMSM, Periodic and OpenTM are used for monitoring. GMSM reduces CPU utilization by 2.7 % and 2.3 % compared to the periodic and OpenTM monitoring.

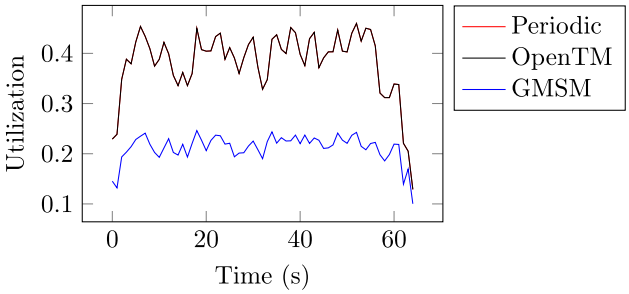

FIGURE 8. Network utilization estimated using GMSM, Period and OpenTM during a 60s interval.

function. The network utilization, $u(t)$, at timestamp t is defined as:

$$u(t) = \frac{\sum link\ capacity}{Total\ network\ capacity}. \tag{2}$$

Fig. 8 shows the estimated network utilization during a 1 min. interval, based on the utilization of the links captured in Fig. 9. Periodic polling (red line in Fig. 9) has the highest estimation accuracy because it uses all switches. The accuracy of the OpenTM method (black line in Fig. 9) is within 6.38×10^{-6} units of the periodic polling method on average. The red time series closely overlaps the black time series. The OpenTM monitoring scheme only ignores those switches whose flow tables are empty and so the difference is small.

A more detailed analysis of the difference is presented in Fig. 9, which illustrates the utilization of every link in Fig. 3 for a 60 second period during one experimental trial. In this scenario seven links, $\{v_5, v_6\}$, $\{v_{10}, v_9\}$, $\{v_{11}, v_{12}\}$, $\{v_5, v_{11}\}$, $\{v_4, v_5\}$, $\{v_1, v_3\}$ and $\{v_3, v_4\}$, are not carrying ICMP, VoIP or VLC traffic but still carry some periodic Link Layer Discovery Protocol (LLDP) packets. These links are monitored by periodic polling but not by the OpenTM approach. The time series resulting from the OpenTM approach are a close approximation of the time series for the periodic polling approach. The bandwidth consumption of the unused seven links are illustrated in subfigures (f) to (I) in Fig. 9. These links are not used by the routing algorithm to deliver traffic. LLDP packets are generated periodically by the OpenFlow Discovery Module. The difference between these seven time series arises due to the way that the OpenFlow Discovery Module issues discover packets. The utilized bandwidth of the seven links

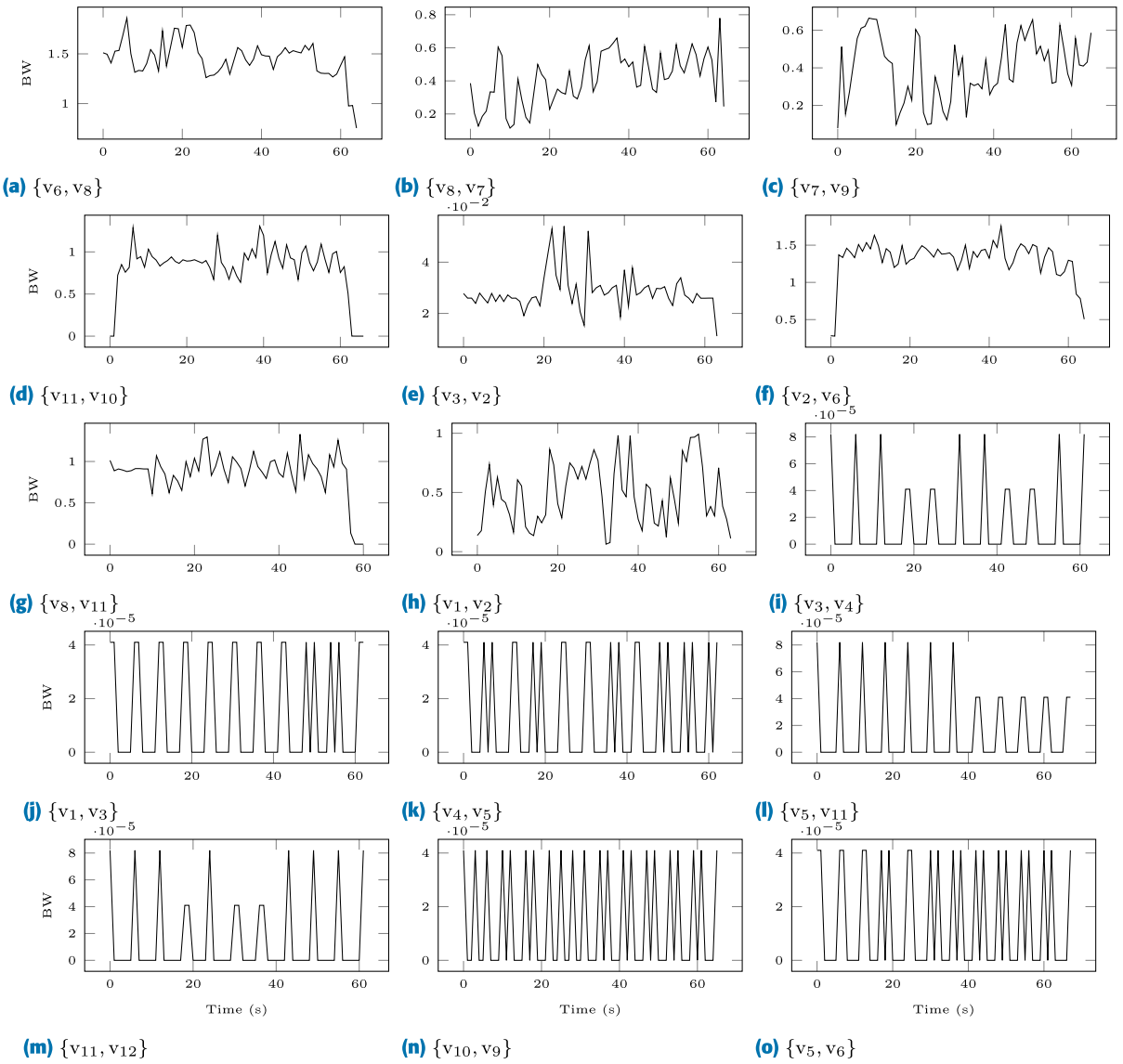

FIGURE 9. Abilene links utilization, which is estimated using bandwidth (Mbps), for 60 s, when the three types of traffic are in operation.

that are not used is small, which accounts for the difference between the OpenTM and periodic polling methods and also the size of this difference. Links that carry more than one RTP flow have a higher BW utilization compared to links which carry a smaller number of RTP flows, or no RTP flows. Subfigures (f) to (I) in Fig. 3 are illustrated to demonstrate the short-comings of the periodic approach: monitoring links that do not carry traffic classes relevant to the network manager offers little utility, particularly when monitoring imposes additional overhead and CPU usage burdens. Regarding the GMSM monitoring approach, the resulting time series are similar to those of a sub-set of links, $\{v_2, v_6\}$, $\{v_6, v_8\}$, $\{v_8, v_7\}$, the links that are involved in GMSM monitoring. Consequently, the bandwidth of 3 links (4 switches) are estimated by GMSM. GMSM does not have a global view about the bandwidth utilization of the entired network. This behaviour is expected as the network manager controls the type of monitoring performed.

GMSM's network utilization estimate has the lowest accuracy. The average difference in accuracy between GMSM and period polling and the OpenTM method are 0.168459 and 0.168452 units respectively. In summary, GMSM provides a monitoring overhead reduction which comes at the cost of accuracy in the network utilization. Given that the user defines what critically important traffic is, via a threshold parameter, they also get to decide what level of information about all traffic on the network via a utilization score is sufficient. The ability to be able to specify and justify what is critically important via a sensitivity threshold allows the network manager to determine the type of monitoring they want, and the loss in accuracy that they can tolerate for a given CoS sensitivity.

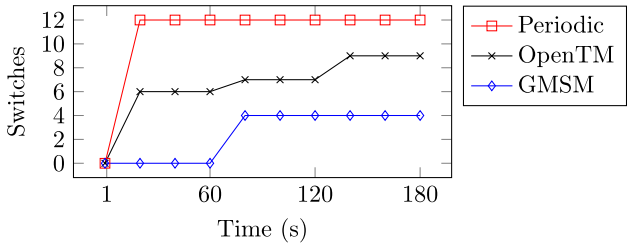

FIGURE 10. The number of switches participating in monitoring is illustrated as time evolves. GMSM queries the smallest number of switches compared to periodic and OpenTM monitoring.

C. DISCUSSION

We present an analysis of the experimental results obtained from the different monitoring schemes compared in this paper. Additionally, we propose use cases where GMSM is an appropriate monitoring solution. GMSM outperformed the periodic and OpenTM monitoring schemes in terms of reducing the overhead and CPU utilization. This reduction resulted from the way that GMSM minimized the number of switches that were included in the monitoring process. Only links that met the threshold were considered. Fig. 10 shows the number of switches that are monitored by each method. GMSM outperformed the periodic polling and OpenTM approaches.

The main disadvantage of GMSM is that it provides the controller with less information compared to the periodic and OpenTM schemes. This impacts the capability of network analysis functions that use the data resulting from monitoring. One benefit of GMSM is that it gives the network operator the ability to be able to select what types of events should be included in the monitoring process. Therefore, they can setup GMSM in a way that makes sure that all the required information about the events of interest will be reported and analysed with high accuracy. In term of applicability, there are a number of use case scenarios where GMSM is a suitable solution. For instance, GMSM can be used as part of a system that enhances QoS while maintaining low overhead, in scenarios where the network operators are familiar with the classes of sensitive traffic being transmitted over their networks. Two examples where this is the case are (i) a multi-cost routing strategies that deal with the problem of meeting the service level agreements of different classes of tenant [36] and (ii) microgrid integration systems, where control system packets are delivered over a shared communications medium and have to compete with other traffic in home networks [37]. It is worth mentioning that the current implementation of GMSM follows the out-of-band SDN architecture. Hence, comparing GMSM with in-band approaches such those in [38] is not a valid case. Similarly, there are a wide range of network modelling and monitoring applications using machine learning techniques such as those presented in [39]. We will consider some machine learning techniques to improve the traffic classification of GMSM towards further enhancement.

Finally, in environments where the network scale is large, techniques like periodic polling could cause great overhead on the network controller [40]. In this type of network a better approach is to specify the events of interest and to have GMSM filter-out what needs to be monitored rather than monitoring all possible events.

VI. CONCLUSION

In this paper, we presented an SDN statistics monitoring system that exposed a traffic CoS sensitivity threshold to the network manager via a graphical visualization called GMSM. It differs from fixed polling and OpenTM monitoring methods by allowing the network administrator to determine the events of monitoring interest. A prototype was designed, implemented and evaluated using simulation experiments on a real-world network topology. Subsequently, we used two metrics to assess the efficacy of the proposed approach. The first metric used assessed the SDN controller overhead, by estimating the frequency of messages exchanged between the controller and the data plane switches, as well as the level of utilization of the Controller's CPU. The second metric used to evaluate GMSM assessed its ability to estimate links utilization. This metric served as a criterion for assessing the accuracy of network analysis.

Experimental findings showed that GMSM lowered the overhead by reducing the number of switches that participated in the monitoring process. This reduction can be interpreted as follows. GMSM concentrated on switches that facilitated crucial network traffic. The network manager determined the level of criticality for various types of traffic based on a set sensitivity threshold. One implication of GMSM's monitoring approach is that it reduced the controller-switch communication by 5.7 % and 6.7 % compared to OpenTM and periodic monitoring. A second implication of GMSM's approach was that the CPU utilization was reduced by 2.7 % and 2.3 % compared to the periodic and OpenTM monitoring approaches respectively. Our findings also indicated that compared to GMSM, the average difference in links utilization accuracy were 0.168459 and 0.168452 units for the periodic and OpenTM approaches respectively. GMSM successfully achieved an overhead reduction and a trade-off was observed in terms of accuracy of utilization estimates, which represents a limitation of the monitoring scheme.

GMSM demonstrates considerable promise, therefore, we will explore the levels of monitoring granularity possible via GMSM, by classifying additional CoS such as online games in future work and by incorporating machine learning to adapt polling intervals. Additionally, we will evaluate the deployment of GMSM in environments where the reduction of monitoring overhead is the crucial requirement for network managers.

REFERENCES

[1] R. de Fréin, C. Olariu, Y. Song, R. Brennan, P. McDonagh, A. Hava, C. Thorpe, J. Murphy, L. Murphy, and P. French, "Integration of QoS metrics, rules and semantic uplift for advanced IPTV monitoring," *J. Netw. Syst. Manage.*, vol. 23, no. 3, pp. 673–708, Jul. 2015, doi: 10.1007/s10922-014-9313-9.

[2] Y. Zhou, J. Bi, T. Yang, K. Gao, J. Cao, D. Zhang, Y. Wang, and C. Zhang, "HyperSight: Towards scalable, high-coverage, and dynamic network monitoring queries," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1147–1160, Jun. 2020.
[3] R. de Fréin, "State acquisition in computer networks," in *Proc. IFIP Netw.*, 2018, pp. 1–9.
[4] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, document RFC3550, 2003.
[5] B. Claise, G. Sadasivan, V. Valluri, and M. Djernaes, *Cisco Systems NetFlow Services Export Version 9*, document RFC 3954 (Informational), Oct. 2004.
[6] (2003). *Making the Network Visible*. [Online]. Available: https://sflow.org/index.php
[7] A. C. Myers, "JFlow: Practical mostly-static information flow control," in *Proc. 26th ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang.*, Jan. 1999, pp. 228–241.
[8] A. Malik, R. de Fréin, M. Al-Zeyadi, and J. Andreu-Perez, "Intelligent SDN traffic classification using deep learning: Deep-SDN," in *Proc. 2nd Int. Conf. Comput. Commun. Internet (ICCCI)*, Jun. 2020, pp. 9–184.
[9] P.-W. Tsai, C.-W. Tsai, C.-W. Hsu, and C.-S. Yang, "Network monitoring in software-defined networking: A review," *IEEE Syst. J.*, vol. 12, no. 4, pp. 3958–3969, Dec. 2018.
[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.
[11] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring network utilization with zero measurement cost," in *Proc. Int. Conf. Passive Act. Netw. Meas.* Cham, Switzerland: Springer, 2013, pp. 31–41.
[12] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for OpenFlow networks," in *Proc. Int. Conf. Passive Act. Netw. Meas.* Cham, Switzerland: Springer, 2010, pp. 10–201.
[13] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "CeMon: A cost-effective flow monitoring system in software defined networks," *Comp. Net.*, vol. 92, pp. 101–115, Dec. 2015.
[14] Q. He, X. Wang, and M. Huang, "OpenFlow-based low-overhead and high-accuracy SDN measurement framework," *Trans. Emerg. Telecommun. Technol.*, vol. 29, no. 2, p. e3263, Feb. 2018.
[15] S. R. Chowdhury, Md. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for software defined networks," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–9.
[16] H. Tahaei, R. Salleh, S. Khan, R. Izard, K.-K.-R. Choo, and N. B. Anuar, "A multi-objective software defined network traffic measurement," *Measurement*, vol. 95, pp. 317–327, Jan. 2017.
[17] H. Xu, Z. Yu, C. Qian, X.-Y. Li, and Z. Liu, "Minimizing flow statistics collection cost of SDN using wildcard requests," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, May 2017, pp. 1–9.
[18] E. F. Castillo, O. M. C. Rendon, A. Ordonez, and L. Z. Granville, "IPro: An approach for intelligent SDN monitoring," *Comput. Netw.*, vol. 170, Apr. 2020, Art. no. 107108.
[19] B.-H. Oh, S. Vural, and N. Wang, "A lightweight scheme of active-port-aware monitoring in software-defined networks," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 3, pp. 2888–2901, Sep. 2021.
[20] K. B. Nougnanke and Y. Labit, "Novel adaptive data collection based on a confidence index in SDN," in *Proc. IEEE 17th Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2020, pp. 1–6.
[21] B. Yan, Q. Liu, J. Shen, D. Liang, and X. Liu, "Cost-effective and accurate flow statistics collection in OpenFlow-based SDN," *Int. J. Netw. Manage.*, vol. 32, no. 4, p. e2197, Jul. 2022.
[22] H. Yahyaoui and M. F. Zhani, "On providing low-cost flow monitoring for SDN networks," in *Proc. IEEE 9th Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2020, pp. 1–6.
[23] W. Watankeesuntorn, K. Takahashi, C. Nakasan, K. Ichikawa, and H. Iida, "Opimon: A transparent, low-overhead monitoring system for OpenFlow networks," *IEICE Trans. Commun.*, vol. 105-B, no. 4, pp. 93–485, 2021.
[24] P. H. Isolani, J. A. Wickboldt, C. B. Both, J. Rochol, and L. Z. Granville, "Interactive monitoring, visualization, and configuration of OpenFlow-based SDN," in *Proc. IFIP/IEEE IM*, 2015, pp. 15–207.
[25] S. U. Rehman, W.-C. Song, and M. Kang, "Network-wide traffic visibility in OF@TEIN SDN testbed using sFlow," in *Proc. 16th Asia–Pacific Netw. Oper. Manage. Symp.*, Sep. 2014, pp. 1–6.
[26] K. Ramarao, "A simple variant of node connectivity is NP-complete," *Int. J. Comp. Math.*, vol. 20, nos. 3–4, pp. 51–245, 1986.
[27] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," in *Proc. 9th Central Eastern Eur. Softw. Eng. Conf. Russia*, Oct. 2013, p. 1.
[28] *POX*. Accessed: Apr. 6, 2023. [Online]. Available: https://github.com/noxrepo/pox
[29] A. Hagberg, P. Swart, and D. S. Chult, "Exploring network structure, dynamics, and function using NetworkX," Los Alamos Nat. Lab. (LANL), Los Alamos, NM, USA, Tech. Rep. LA-UR-08-05495; LA-UR-08-5495, Jan. 2008.
[30] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0–survivable network design library," *Netw., Int. J.*, vol. 55, no. 3, pp. 86–276, 2010.
[31] (2014). *SIPp*. [Online]. Available: http://sipp.sourceforge.net/
[32] A. Botta, A. Dainotti, and A. Pescapé, "A tool for the generation of realistic network workload for emerging networking scenarios," *Comp. Net.*, vol. 56, no. 15, pp. 3531–3547, 2012.
[33] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw.*, Oct. 2010, pp. 1–6.
[34] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.
[35] O. Izima, R. de Fréin, and A. Malik, "A survey of machine learning techniques for video quality prediction from quality of delivery metrics," *Electronics*, vol. 10, no. 22, p. 2851, Nov. 2021.
[36] A. Malik and R. de Fréin, "SLA-aware routing strategy for multi-tenant software-defined networks," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2020, pp. 1–7.
[37] Y. Emir Kutlu, R. de Fréin, M. Basu, and A. Malik, "Integrated DC microgrid testbed for QoS evaluation," in *Proc. Joint Eur. Conf. Netw. Commun. 6G Summit (EuCNC/6G Summit)*, 2023, pp. 1–2.
[38] I. I. Awan, N. Shah, M. Imran, M. Shoaib, and N. Saeed, "An improved mechanism for flow rule installation in-band SDN," *J. Syst. Archit.*, vol. 96, pp. 1–19, Jun. 2019.
[39] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, "RouteNet: Leveraging graph neural networks for network modeling and optimization in SDN," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2260–2270, Oct. 2020.
[40] B. Huang and S. Dong, "An enhanced scheduling framework for elephant flows in SDN-based data center networks," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2020, pp. 1–7.

ALI MALIK received the Ph.D. degree in computing from the University of Portsmouth, U.K., in 2019. He is currently a Postdoctoral Researcher with the School of Electrical and Electronic Engineering, Technological University Dublin, Ireland. His current research interests include SDNs, VANET, traffic engineering, ML, and cybersecurity.

RUAIRÍ DE FRÉIN received the B.E. degree in electronic engineering and the Ph.D. degree in time-frequency analysis and matrix factorization from University College Dublin (UCD), Ireland, in 2004 and 2010, respectively. He held a Marie Skdowska-Curie Fellowship with the KTH Royal Institute of Technology, Stockholm, and also with Amadeus SAS, Sophia Antipolis, France. He is currently a CONNECT Funded Investigator and a Lecturer with the School of Electrical and Electronic Engineering, Technological University Dublin, Ireland. Over the past few years, he has developed algorithms for predicting quality-of-delivery metrics for network management and monitoring strategies for small cell networks, and monitoring techniques for internet protocol television (IPTV). His research interests include machine learning for integrating microgrids, sparse signal processing, software-defined networks, and VANETs.