Received November 13, 2017, accepted December 27, 2017, date of publication January 3, 2018,
date of current version February 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2017.2789281

Cost Effective Network Flow Measurement for Software Defined Networks: A Distributed Controller Scenario

HAMID TAHAEI[1], **ROSLI BIN SALLEH**[1], **MOHD FAIZAL AB RAZAK**[1,2], **KWANGMAN KO**[3], AND **NOR BADRUL ANUAR**[1]
[1]Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia
[2]Faculty of Computer Systems & Software Engineering, University Malaysia Pahang, Pahang 26600, Malaysia
[3]Department of Computer Engineering, Sangji University, Wonju 220-702, South Korea

Corresponding authors: Hamid Tahaei (hamid.tahaei@siswa.mu.edu.my) and Nor Badrul Anuar (badrul@um.edu.my)

This work was supported in part by the collaboration between Universiti Malaya and Institute for Information & Communication Technology Promotion Grant through the Korea Government under Grant MSIP 2017-0-01705 and in part by the Basic Science Research Program through the National Research Foundation of Korea, Ministry of Science, ICT, and Future Planning, under Grant 2017030223.

ABSTRACT Software-defined networking (SDN) has emerged as an evolutionary paradigm in datacenter networks by separating data from control plane and centralizing network decision making. Traffic flow measurement in SDN is relatively lightweight in comparison to the traditional methods. It enables flow measurement system to overcome the issues of traditional measurement systems, such as cost and accuracy by employing a centralized controller. Nevertheless, a full physically centralized controller introduces negative impacts on the network as well as the measurement system (i.e., introducing extra overhead or accuracy issues). However, few efforts have been devoted to measurement techniques in SDN distributed controller architecture, where every controller pulls its corresponding flow statistics, and these statistics are required to expose by only one single expression as if they are collected by one controller. Moreover, the imposed costs of flow measurement in distributed controller architecture are still an issue that remains unsolved. In this paper, we attempt to fill in this gap and present a novel and a practical solution for a cost-effective measurement system in SDN distributed controller deployment. We also propose a synchronization mechanism for aggregating traffic statistics in the multiple controller model. We evaluate our method through extensive emulations in a datacenter topology and present our findings to demonstrate the impact of multiple controllers on overhead and accuracy.

INDEX TERMS Software defined networking, network traffic monitoring, network flow measurement, flow statistical collection.

I. INTRODUCTION

Flow measurement system is an essential requirement for network applications. Many applications in a Datacenter Network (DCN) such as, anomaly detection, network planning, billing, load-balancing, traffic engineering and security require an accurate and timely-basis flow measurement system for monitoring and recording the flows of network traffic [1]. Traditional flow measurement systems, such as NetFlow [2] and sFlow [3], apply packet sampling approaches to collect information about packets in the network and analyze this information to infer flow level statistical measurement. They have either a low accuracy or a high deployment cost and consume more resources [4]. An example of the later is the deployment of NetFlow, which requires setting up of collectors and analyzers. Moreover, enabling NetFlow in the routers may degrade the packet forwarding performance [5]. Furthermore, NetFlow and similar tools such as Sflow, Jflow, IPFIX, and PRTG are hardware-based feature that need to be configured to be set for each individual interface on the physical device (switch/router). In reducing the limitation in the traditional flow management systems, the most recent measurement methods alleviate the accuracy and cost issues by applying the emerging technology known as Software Defined Networking (SDN).

In SDN, a central controller collects flow statistics by either directly requesting from switches (pull-based approach) or passively receiving them from switches (push-based approach) upon the expiration of their corresponding flows. The statistics reach the central controller to be used by on-demand applications (i.e., routing and load balancing) in the network. Thus, eliminating the sophisticated process of sampling approach for flow level measurement used in traditional methods results in more accurate and near real-time flow measurement by actively collecting flow's information from switches. However, both of the aforementioned approaches in SDN lead to a massive cost of the controller's channel bandwidth and processing delay for a single SDN controller [6]. The situation becomes worse in in-band SDN network deployment when monitoring and routing traffic shares bandwidth along the same link. Resulting in a delay of flow statistics to be reached at the central controller as normal network traffic disturbs the measurement traffic and the flow statistics. In addition, like any other centralized system, a fully physically centralized controller is inadequate and introduces issues of scalability, reliability and performance bottleneck [7]. To overcome these obstacles above, industry and academia proposed multiple SDN controller designs by which the central controller can be physically distributed but logically centralized [8].

Several published papers have explored various techniques to cope with the limitations of the flow measurement system (i.e., the tradeoff between accuracy and overhead) in fully physically centralized SDN controller, however, proposing a cost-effective with accuracy design which can be implemented in multiple SDN controller yet remained intact. For example, in the single SDN controller, OpenTM [9] utilizes pull-based strategy to collect a single flow statistic in every request. It reported highly accurate flow statistical measurement, however, applying per-flow query strategy introduces communication overhead if the number of flows in the network is large. Payless [10] proposed an adaptive pull-based method to overcome the overhead on OpenTM, However, an extra communication overhead and message interactions cannot be neglected if the traffic spike is high for every flow. In contrast, FlowSense [11] applies a push-based approach to infer link utilization based on passive capturing of flow arrival and expiration messages which incur zero overhead. However, the link utilization can only be calculated at the discrete points of time upon the expiration of the flow. Thus, this is unable to fulfill the dynamic requirement, and moreover, accuracy of the results is untrusted (uninsured). CeMon [12] proposed a pull-based approach combining the single flow request and aggregating all flow statistics in a single flow request which can be implemented in multiple SDN controller. However, it applies a greedy optimization algorithm to select the target switches to pull which causes degradation of accuracy. Moreover, due to the complexity of the algorithm, it is unreliable for a large scale of networks with a large number of flows. Furthermore, the overhead generated by the synchronization of multiple controllers was unrevealed.

To address the absence of traffic measurement system in the distributed SDN controller design and overcome the primary challenge of the flow measurement system (i.e., minimizing overhead), this paper proposes an accurate and cost-effective strategy for near real-time flow measurement system in DCN. By ''cost effective'', the paper implies multi-objective overheads in terms of traffic volume generated for measurement purpose, message interaction and controller overhead as well as the CPU utilization. The main contribution of this paper is three folds. First, we reveal a significant strike on the cost of flow measurement by minimizing various overheads in a single controller design. Second, we propose a generic framework for flow measurement in datacenter which applies in both single and multiple-controller and show the effectiveness of the proposed method under different measurement scenarios. Lastly, we formulate a cost-effective multi-objective controller design in in-band network deployment and manifest the performance evaluation of the proposed multi-objective controller design and the state-of-the-art approaches through emulation.

The proposed architecture utilizes local controllers to pull flow statistic and forwards statistics to an upper layer application to aggregate all the counters and shaping a universal flow measurement in the network. We designed a coordinator level on top of all the controllers connecting to the switches. We apply group feature introduced by OpenFlow 1.3 which wildcards all the demanded flows in a single group then utilize aggregated pulling request to collect statistics. The proposed design is implemented as a standard northbound interface which can utilize both fixed and adaptive pulling systems. This work is an extend of our previous attempted in [13] where we proposed an active measurement method in a single SDN controller scenario with out-of-band deployment. To the best of our knowledge, this is the first attempt on SDN that proposes traffic measurement for distributed controller in in-band network deployment which uses emulation as an experiment.

The rest of this paper is organized as follows. Sections 2 presents related work in SDN traffic measurement and discloses the challenges. It also reviews related works on multi-controller Environment. Section 3 presents a brief background of native measurement approaches in SDN and outlines the system design. The section is further followed by elaborating the problem formulation. Section 4 explains the proposed optimal solution followed by a heuristic case study to analyze the proposed system behavior. Section 5 explains a multi-objective optimization for multi-controller scenario in in-band network deployment. In section 6, the proposed architecture is evaluated under extensive experiments with different controller number. Finally, we conclude the paper in Section 7.

II. RELATED WORK

Traditionally, flow-based network measurement tools have been introduced in traditional IP networks. NetFlow [2] from Cisco is the premier and the most prevalent, uses a central collector to analyze the sampled or complete traffic statistic. It supports various technologies such as Multi-cast IPSEC and MPLS. Later by releasing version 9, it became a universal standard by IP Flow Information Export (IPFIX) IETF working group. Using this standard, Cisco NetFlow collector can be used by non-Cisco devices. InMon introduced sFlow [3] which uses time-based packet sampling for capturing flow-based IP traffic. Similar to NetFlow and sFlow, Jflow [14] proposed by Juniper Networks, also exploits statistical sampling to analyze flows and monitor detail information about flows. However, as mentioned earlier in the introduction; these tools are all hardware-based features which require to be enabled in each individual device and interface. In addition, all of these monitoring tools mentioned above are commercialized and incur licensing. Furthermore, none of these tools can be applied in either distributed or single SDN controller and require investment and cost for deployment in the network.

A. NETWORK MEASUREMENT IN SDN

SDN Network measurement has been a subject of recent academic topics. Several efforts in single controller have been observed that proposed different methods in single SDN controller design with various QoS requirements to overwhelm challenges associated with accuracy and overheads. These methods are broadly decomposed into two main categories (i.e., active and passive).

In the active measurement approach, a probe packet is continuously sent over network paths as a request to collect flow statistics. Such methods offer different level of granularity, resulting in the highly accurate flow measurement. As such; imposing significant measurement overhead that may disturb the critical traffic flows. The situation becomes worse in in-band network deployment due to using same links' bandwidth in routing traffic and the generated traffic by monitoring purpose. The active measurement demands careful planning to cope with the requirements of a centralized control architecture in SDN. Deploying active measurement considerably increases the data acquisition and results in the centralized control mechanism of the SDN hitting saturation [15]. Even applying fast analytical models and statistical models (e.g. Machine learning techniques and Monte Carlo) at the end-to-end QoS measurement, the controller may face challenges relating to bottlenecks in communications, control and optimization [16]. OpenTM [9], a traffic matrix estimation system is proposed to get flow statistics using simple logic for querying flow table counters with different querying strategies. Such a mechanism gathers active flow statistics on a one-by-one which is considered to be costly in terms of introducing communication overhead in the network. Chowdhury *et al.* [10] proposed an adaptive statistical collection algorithm, which emphasis on the tradeoff between accuracy and network overhead. This approach has a low overhead and achieves a higher accuracy of statistical collection by capturing traffic spikes. The work in [13], presents an active approach for elastic and fixed pulling switches and highlighted the tradeoff between accuracy and overhead. However, it is proposed for single controller with out-of-band deployment. In a similar vein, CeMon [12] proposed a low-cost monitoring system which adaptively pulls switches for statistical collection and optimizes the pulling cost for all active flows. However, the proposed method may result in loss of accuracy in different topology as the greedy switch selection algorithm is highly reliant on the behavior of flows in the network. Also, it is scenario dependent which is suitable for networks with a small rate of new flow arrival. Moreover, CeMon implemented its proposed method for distributed controller model without taking the factor of overhead for synchronization of multiple controller.

In contrast, in passive measurement; real-time traffic is captured and analyzed at the predefined points of the network. In this approach, the network is manually captured and its traffic is directed to an analyzer or agent for further processing. Since there is no probe packet in passive measurement; therefore, it does not cause any overhead. Such method allows for the processing of local traffic states, and global behavior of the network traffic flows passing a specific network point. Passive measurement methods are considered non-intrusive and it does not generate extra traffic in the SDN, but these methods need packet-sampling and statistical methods to conclude the state of the network traffic. Two key limitations of these techniques are (1) Inaccurate measurement; because small flows being missed or multiple monitoring nodes samples the similar packet [17], and (2) the necessity for a complicated analytical mechanism to process network traffic at the high speed in DCNs. For example, Flowsense [11] presented a push-based technique which with no overhead while measuring the network link utilization. However, this method gets the link utilization at discrete points in time with a lengthy delay; which results in losing the accuracy of the flow statistics.

B. DISTRIBUTED CONTROLLER

Deploying distributed controller in SDN has been proposed to address the issues of scalability and reliability that a single controller suffers from [8]. However, the concept of distributed control plane in SDN implies the multiplicity of the physical entity of the control plane where it addresses some obstacles such as limited scalability, reliability and being a single point of failure.

Till now, several deployment models have been proposed for distributed control plane with having various objectives and requirements. For example, to address the aforementioned challenges, a model has been proposed which is referred to as physically distributed but logically centralized. To meet the requirement of centralized view of the network, controllers such as Onix [18], Hyperflow [19], ONOS [20], and OpenDayLight [21] share information among each other to have a consistent view of the network. However, to provide a full visibility (global view of the network), controllers have to constantly synchronize their state with each other. This synchronization may cause network overhead as the network state frequently change and controller should constantly be synchronized. It has been fully understood that an inconsistent control state of the network can negatively affect many application's performance as well as the reliability of control plane [8]. To address this issue, a fully distributed model was introduced where the control plane is both physically and logically distributed. Kandoo [22] proposed a model based on fully distributed deployment known as hierarchical deployment model where employs two levels of controllers such that a root controller takes a full visibility of the network and controls all the local controllers in the second level. All the low-level controllers (local controllers) orchestrate their own domain and synchronize their domain states with the root controller. Cluster deployment is another model which employs multiple controllers with the same role. However, according to OpenFlow [23], at most one controller in master state can control a switch. Hence, the backup controller may take the control of switch(s) (become the master controller) on the failure of a master controller. Although, very minor efforts have been done on network measurement methods for distributed controller model, no published attempt has gone on striking the problem of minimizing different costs (overheads on communication, messaging and controller) of active measurement in the real setting of distributed SDN control plane.

III. SYSTEM DESIGN

Our aim is to estimate the traffic volume of a set of arbitrary flows in datacenter environments. In this section, we first provide a general background of OpenFlow and then explain our proposed design with further formulating our research problem.

A. BACKGROUND

In OpenFlow, the monitoring task is accomplished by a controller, which is connected to the switches via a secure channel interface called southbound interface. The secure channel is established over a TCP connection between the controller and the switch. The controller accumulates the real-time flow statistics from the corresponding switches, and combines the raw data to deliver interfaces for upper-layer applications. When a switch receives the first packet of a new flow in the network, it first checks its flow table to find a match for the flow. Then the flow is forwarded based on the corresponding flow entry in the flow table. In the case of table miss (when there is no match for flow); the switch forwards the first packet header to the OpenFlow controller by a packet-In message. The controller processes the packet header and makes further actions such as setting up the routing paths. The controller then instructs the corresponding switches along the path by a packet_out message.

According to OpenFlow specification 1.0 [24], a naive approach to obtain a specific flow statistic in the network is to query it from the switch using OpenFlow single stat-request (SSR). In this way, fine-grained per-flow information about a predefined individual active flow is requested with the "ofp_flow_stats_request" request type. The predefined active flow is queried based on the exact match of several fields such as input port, source/destination address, and so on. Thus, for measuring byte count of each active flow, the controller sends one request message to the switches and receives one reply message in response (two messages for each flow). Another approach is known as pulling all (PA), queries all the active flows from the switch(s). In this way, per-flow statistics of all the active flows in a switch are collected (aggregated) in a single file and sent to the controller regardless of any match. Later in the specification1.5.0, OpenFlow, introduces "Flow Entry Statistics Trigger" (FEST) which is a mechanism that automatically sends Flow entry thresholds statistics to the controller base on various statistics thresholds (a list of statistic field thresholds). However, FEST, is unsuitable for timely-basis flow measurement as it reports flow statistics based on the predefined thresholds regardless of measurement intervals

B. IN-BAND AND OUT-OF-BAND NETWORK DEPLOYMENT

The deployment of SDN in network falls into two categories: (1) out-of-band and (2) in-band deployment. In out-of-band, signaling requires a dedicated network between the controllers and switches, which incurs an additional cost in terms of hardware infrastructure. However, this deployment provides much better response time and does not suffer from potential congestions [25]. Therefore, the overhead costs for this deployment depend on the number of flow which is pulled from the switch regardless of controller placement.

In in-band deployment, transmission of control message and data message take place in a shared network bandwidth [26]. This incurs no additional cost for inter-controller communication [25]. The measurement and data traffic shares bandwidth, proactively fetching counters with high frequency notably impact the efficiency of data transmission. Also, it is relatively complex to compute the communication cost compare with the out-of-band deployment. Therefore, the hops from the pulling switch to the controller should be taken into account. Though, costs for message interaction and controller overhead remain the same for both out-of-band and in-band deployment.

C. ARCHITECTURE

The architecture of the propose design consists of two stages as follows: (1) a local controller design which describes the entire schema in a layout like steps and (2) a core design which focuses on the design of the local controllers.

1) DESIGN OF LAYOUTS

Fig. 1 depicts the schema of the proposed system layout. In general, it consists of three layers: OpenFlow network Layer, OpenFlow controller level and a coordinator level on top of all the controllers connecting to the switches.

OpenFlow network level consists of all the Low-level network entities such as hardware and software devices connected to the upper layer via the northbound interface.

The controller level is the heart of our design where statistics of flows are collected by their associated local controller in each time interval (1 second in our design). Each controller

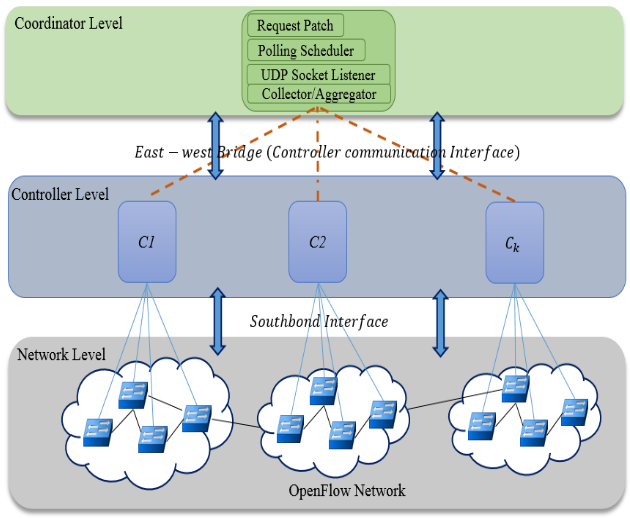


FIGURE 1. **Schema of system layout.**

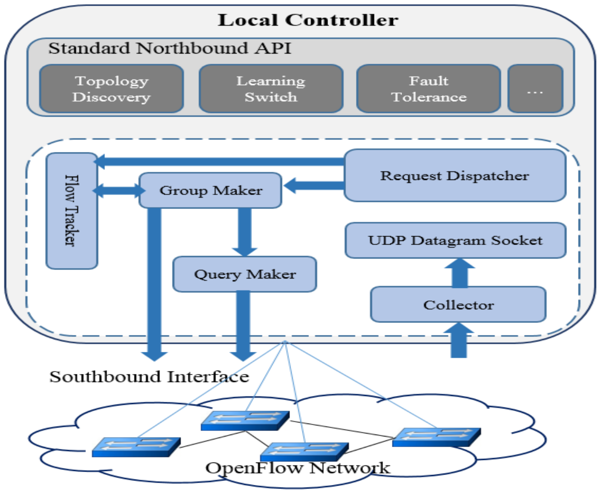


FIGURE 2. **Local controller.**

which is associated with a flow (tracking and instructing flow) is set by the top layer to request the flow statistics and responsible to collect, aggregate, and forwarding them to the upper layer (i.e., coordinator).

The coordinator level is responsible to set controller(s) to request flow statistics (Request Patch) in an arbitrarily fashion (fixed or adaptive pulling) using "Pulling scheduler" and receive statistics from different controllers by "UDP Socket Listener" and accumulates them to shape a traffic matrix (TM) of demanded flows. This layer provides an East-west interface to interconnect controllers and bridging all the gathered statistics to accumulate the demanded task. The East-west interface is often so-called east-west bridge where it is responsible to implement efficient communication, synchronization and negotiation function among multiple controllers [8].

2) LOCAL CONTROLLER DESIGN

Fig. 2 shows the architecture of a local controller. There are four steps to accomplish the measurement task after a local controller receives flow statistic request from the coordinator using "Request Dispatcher" module.

Flow Tracker: The first step is tracking all the flows (current/new flows in the controller domain) with a specific characteristic (user demands) which is required to be monetarized.

Group Maker: The second step is grouping all the flows which were specified earlier in the first step. This module utilizes "group table" feature in OpenFlow specification 1.3. It then instructs the switch(s) to modify the associated TCAM output group entry by sending a packet_out message to switch(s).

Query Maker: In the third step, switches are pulled with the exact match of the created group in the previous step.

Collector: All the statistical counters in each time intervals are aggregated by this module and sent to the top layer (coordinator). The process of sending aggregated stats is performed by a simple UDP datagram socket.

We implemented the proposed design as a northbound application on top of the controllers. The coordinator can access multiple local controllers by a simple application call. The proposed design can be implemented in various single and multiple controller scenario such as clustered, distributed, and hierarchical. Furthermore, it is able to accomplish almost all aspects of a monitoring system (such as flow utilization, measuring available bandwidth, packet loss, link and packet delay, and so on).

D. PROBLEM DEFINISTION

As mentioned earlier, there are three approaches for collecting traffic stats such as (1) single stat-request (SSR), (2) wildcarding all fields to collect all flows (pulling all), and (3) "Flow Entry Statistics Trigger" (FEST).

SSR generates two messages to collect statistics of a flow in each interval, four messages for two flows and so forth. The main drawback of this approach is the imposed overhead for generating a huge number of request and reply message in the network which also utilizes CPU cycle of network devices as well as SDN controller.

PA collects all the active flows statics in a switch by only two messages: request and reply messages. This significantly reduce the number of message interaction and communication cost as well as repeated reply headers for a high number of flows. However, excessively applying the second approach causes flow statistics overlapping which imposes extra message interaction overhead and communication cost as well as an overhead in the controller. Another drawback of PA is the lack of control on flows query as it pulls all the active flows in the network regardless of the actual need.

In the latest approach (FEST), OpenFlow enabled switch automatically sends flows' statistics to the controller when one of the pre-defined threshold(s) is triggered. The drawback of this approach is that many active flows are lost when the defined threshold is not triggered if they are below the threshold. In addition, setting the least value of threshold (i.e. one bytes) can be a bottleneck for the CPU utilization of the device, resulting in reporting the stats with delay. However, FEST only reports statistics upon a flow is triggered by a threshold which is inappropriate for timely basis flow measurement where statistics are collected base on time intervals.

1) COMMUNICATIN COST

According to the OpenFlow specification 1.3 [23], the minimum length of flow stat-request message l_{rq} in wire is 122 bytes [12]. However, this length is highly associated with the flow specification match. For example, the length of a single flow statistic request (SSR) which is specified by a normal 6 tuple fields (i.e., Ethernet type, IP protocol, IPv4 src, IPv4 dest, UDP src port, and udp dst port) is 162 bytes including 66 bytes for packet header (Ethernet + IP + TCP headers) and 96 bytes for packet payload (flow match and instructions). The request length for PA approach is the minimum 122 bytes as there is not specification for matching flows.

We captured the reply message by Wireshark and observed that Reply message header length l_{rp} for SSR and PA is 82 bytes and 162 bytes respectively. The length for each single flow entry l_{sp} stat for both of the mentioned approach above is 144 bytes. However, the reply message may split into multipart messages as the maximum size of a TPC packet is 64bytes in a medium. The length of UDP message[1] [27] containing aggregated statistic sent by every local controller to the coordinator is 60 bytes which is donated by l_{udp}. Therefore, the total communication cost $Cost_{com}(f)$ of SSR and PA for pulling a set of arbitrary flows f for each individual controller in out-of-band deployment is a linear function of f as in equation 1, 2 respectively. As such, equation 3 and 4 describe the linear formulation formulate of communication cost $Cost_{com}(f)$ of SSR and PA in in-band deployment.

a: OUT-OF-BAND

$$Cost_{com}(f) \cong \sum\nolimits_{fi} \left(l_{rqi} + l_{rpi} + l_{sfi} + \left(l_{udp} \times v_{kc}\right)\right), \quad \forall f_i \in f(0 < i < m), \quad f \subseteq F, \; v_{kc} \in v, \; v \subseteq V \tag{1}$$

$$Cost_{com}(f) \cong \left(l_{rq} + l_{rp} + \sum\nolimits_{f_z \in F} l_{sfz}\right) v + (l_{udp} \times v_{kc}), \quad \forall f_z \in F(0 \leq z \leq |F|), \quad v_{kc} \in v, \; v \subseteq V \tag{2}$$

b: IN-BAND

$$Cost_{com}(f) \cong \sum\nolimits_{fi} \left(l_{rqi} + l_{rpi} + l_{sfi} + \left(l_{udp} \times v_{kc}\right)\right) \times h_{\alpha\beta}, \quad \forall f_i \in f(0 < i < m), \quad f \subseteq F, \; v_{kc} \in v, \; v \subseteq V \tag{3}$$

[1]A UDP packet may contain empty datagram (no data). However, the minimum length in the wire over Ethernet is 60 bytes. The minimum elements that contribute in the length of an UDP packet is, Ethernet header, IPv4_header, and UDP_header which equals 14 + 20 + 8 = 42 bytes. However, as per by Linux host driver extra bytes is padded to the packet to full fill the requirement of minimum length of packet in the Ethernet.

TABLE 1. Notation of problem formulation.

Notation	Description
$G = (V, E)$	The target network with V as a set of switches and E as a set of links.
F	Universe flows in the network.
C	A set of Controllers.
n	Message (request/reply/controller-to-coordinator)
l_{rq}	The Length of request message (122 bytes).
l_{rp}	The length of reply message (82 bytes).
l_{sf}	The length of every entry in a file (136 bytes).
l_{udp}	Minimum length of UDP message (header and body 60 bytes).
θ	The number of instruction taken to fragment stat-reply file.
λ	The number of instruction taken to read stat-entry.
K	Coordinator
$h_{\alpha\beta}$	The number of hops (nodes) from switch v_α to controller $h_{\alpha\beta}$

$$Cost_{com}(f) \cong \left(l_{rq} + l_{rp} + \sum\nolimits_{f_z \in F} l_{sfz}\right) \times v + (l_{udp} \times v_{kc}) \times h_{\alpha\beta}, \quad \forall f_z \in F(0 \leq z \leq |F|), \quad v_{kc} \in v, \; v \subseteq V \tag{4}$$

Where in the network graph G = (V, E), V = {v1, v2, . . . , vn} is the set of switches and E represent the set of links between switches, with set V, donated as a set of switches, and $f = \{F : f_i \in F, 0 < i < m\}$ where F = {f1, f2, . . . , fm} is the total current flows (universe) in the network and C = {c1.c2, . . . , ck} a set of controllers Table 1 listed notations of the problem formulation.

2) MESSAGE INTERACTION COST

As the number of flows increase in the network, pulling their statistic becomes more frequent. In other words, the more flows in the network is, the more message is interacted between the controller and switch(s). This makes pull-based approach inefficient for continuous measurement with high-granularity due to consuming too great portion of switch-controller bandwidth as well as switch CPU [28]. Furthermore, Sünnen [29] showed that when the read-stats messages are sent too often, the switch's CPU utilization and the number of Spending messages increases. Thereby, given the network graph G with set of flow f and a set of controllers C, the total number of message interaction for each individual controller in SSR and FA can be found in a linear function of $Cost_{message}(f)$ in equation 5 and 6 respectively. Where n_{rq}, n_{rp} and n_{udp} are "ofp_flow_stats_request", "ofp_flow_stats_reply", and "udp for coordinator" messages respectively.

$$Cost_{message}(f) \cong \sum\nolimits_{fi} \left(n_{rqi} + n_{rpi} + n_{udpi}\right), \quad \forall f_i \in f(0 < i < m), \quad f \subseteq F \tag{5}$$

$$Cost_{message}(f) \cong \sum\nolimits_{vj} \left(n_{rq} + n_{rp} + n_{udp}\right), \quad v_j \in v, \quad v \subseteq V \tag{6}$$

TABLE 2. MIPS assembly instruction language taken by CPU. Adopted from [30].

λ	θ
2 ✕ Unconditional jump (Jump to target address, For switch, procedure return)	Unconditional jump (For procedure call)
Data transfer (Byte from memory to register)	Data transfer (Byte from memory to register)
Conditional branch (Compare less than)	
Arithmetic (subtract)	
Data transfer (Byte from register to memory)	

Subject to:

$$p = \{m|m : N\}, \quad p = \begin{cases} 1, & Cost_{com}(f) \leq 6550\ bytes \\ m \times 2, & Cost_{com}(f) > 6550\ bytes \end{cases}$$

Where the condition stated in equation 3-7 defines a reply message is split into two parts (two messages) if the length is greater than 6550bytes. This is the default packet size in the network cannot be greater than 64Kb for efficient transfer of data in the network.

3) CONTROLLER COST

The controller's overhead implies the utilization of controller's CPU which is also referred to as the computation overhead [31]. It can be defined by the number of instructions imposed by execution, calculation and comparison of raw data to process byte-count (calculating byte count of each flow and subtracting it from previous count). The performance and throughput of CPU is measured from different perspectives such as Cycles Per Instruction (CPI), Million Instruction Per Second (MIPS), and Transaction Per Second (TPS). Thereby, the CPU instruction rate is calculated by dividing the observed CPU cycle speed by the observed CPI [17]. However, determining the exact number of instruction applied by CPU requires obtaining job's information (calculation of statistic reply) comprised of multiple tasks each of which consists of multiple threads, which is out of the scope of this paper. In addition, different CPU generation (32 or 64bit CPU registers) results different performances in various CPU instructions such as Instruction Fetch (IF), Instruction Decoder (ID), Execution (EXE), Memory/IO(MEM), Write-Back (WB) each of which consists of various instructions which increases the clock cycle time [32]. Therefore, analyzing and calculating CPU performance by the number of instructions in practice is not a rational way.

A simple criterion to observe the imposed overhead is to presume a constant value θ and λ indicating the number of instruction taken by CPU for fragmentation of the stat-reply files (data transfer) and reading stat-entry (arithmetic, data transfer, logical, conditional, and jump) respectively. For simplicity, we assume θ (reading stat-file and put into memory) and λ (subtracting the current flow count from the previous one and put into memory) take 2 and 6. Table 2 shows the MIPS assembly instruction language taken by CPU [30]. Therefore, the controller overhead $Cost_{controller}(f)$ in SSR, and PA, analyzing n specific flows from set f in each interval is formulated as a linear functioned of m in equation 7 and 8 respectively.

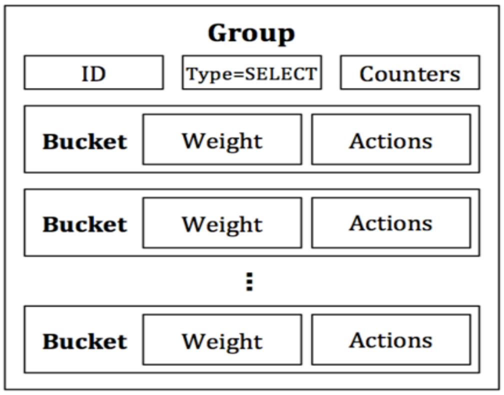


FIGURE 3. The select group [33].

$$Cost_{controller}(f) \cong \sum\nolimits_{fi} (3\theta + \lambda_i), \quad \forall f_i \in f(0 < i < m), \quad f \subseteq F \tag{7}$$

$$Cost_{controller}(f) \cong \sum\nolimits_{vj} \left(3\theta_j + \lambda_i\right), \quad v_j \in v, \ v \subseteq V \tag{8}$$

IV. OPTIMAL SOLUTION

The optimal solution is found through wildcarding the demanding flow set f and subtracting them from the universe flow F. Therefore, the reply message contains only demanding flows in each interval. Thus, reducing all the aforementioned problem above by generating two messages each time interval (i.e., one request message and one reply message). However, according to OpenFlow, flows are wildcarded either by all fields (PA) or "some cases bitmasked" (such as IP-scr, IP-desc and input-port) [23].

We adopt "group table" that is introduced by [23] for traffic Engineering, load balancing and fast failover purposes. A group can either have a single or a list of action/bucket. In general, there are four types of group such as "All", "SELECT", "INDIRECT", and "FAST-FAILOVER", each of which has specific features (interested readers are referred to [23] for more information). We implement "SELECT" type in which each packet entering the group is sent to a single bucket associated with its action. Thus, for this group in a switch we define all potential output actions related to a flow. In such a case, all the incoming flows or current flows are grouped without any intervention to the forwarding decision and central policy enforcement. Therefore, the action of a flow entry is set to the action or a list of actions for that group. We then request the group rather than pulling a single (SSR) or all flows (PA). Fig. 3 shows the SELECT group [33] type in OpenFlow 1.3. The pseudo code to construct group and mapping flows to the group is shown in Fig. 4.

An integrating group table with aggregating demanding active flows in the network provides the feasibility to

***Algorithm 1** mapping flows to group table*

Input: $f = \{x \mid x \subseteq F\}$: set of flows, $G = (V, E)$: the network, δ = match

```
1:  c = create(Group_i)        // creating group on the switch
2:  for each $x \in f$ do
3:      if (isNew(x)) then         // check if the flow is new
4:          if (x[attribute] == δ)
5:              Group_i ← x
6:          end if
7:      end if
8:  end for
9:  return c
```

FIGURE 4. The pseudo code to construct group and mapping flows to the group.

wildcard a set of specific flows for the purpose of fine-grained measurement. In this case, the optimal number of flows in every interval is captured. By implementing our solution, the length of request lrq and reply message lrp are 122 bytes and 218 bytes respectively. However, similar to PA the reply message may split into a multipart reply message if the length exceeds 64Kb in the medium. The optimal solution for communication cost, message interaction, and the controller overhead (for each controller) in the network G with an arbitrary set of flows f can be formulated in equation (9,10,11) in out-of-band network deployment respectively.

$$OptimaCost_{Com}(f) \cong l_{rq} + l_{rp} + \sum\nolimits_{fi} l_{sfi} + (l_{udp} \times v_{kc}), \quad \forall f_i \in f(0 < i < m), \quad f \subseteq F, \quad v_{ck} \in v, \quad v \subseteq V \tag{9}$$

$$OptimaCost_{message}(f) \cong n_{rq} + n_{rp} + n_{udp} \tag{10}$$

$$OptimaCost_{controller}(f) \cong 3\theta + \sum\nolimits_{fi} \lambda_i \quad \forall f_i \in f(0 < i < m), \quad f \subseteq F \tag{11}$$

A. CASE STUDY: SYNTHETIC TOPOLOGY

After describing the problem and a set of equations, this section presents an analysis case study on a single centralized controller to examine the feasibility of the proposed design in out-of-band deployment. The case study comparatively investigates the relationship between cost factors and flows number by which the proposed solution is applied on the synthetic topology and the result is compared. The objective of this case study is to measure the utilization of all flows passing through the link pi. The topology used in this case study is the pod 1 (consists 4 switches, two edges and two aggregation switches) of 4-pod fat-tree which is a common topology used in datacenters (shown in Fig. 5). Maximum universe flow F=2000, initial number of arbitrary flows passing through the path p_i is f = 10, and initial flow number in the node v is 100 with the increase ration of 66,40,28, and 20% in the next 4iterations. The controller C1 pulls the switch

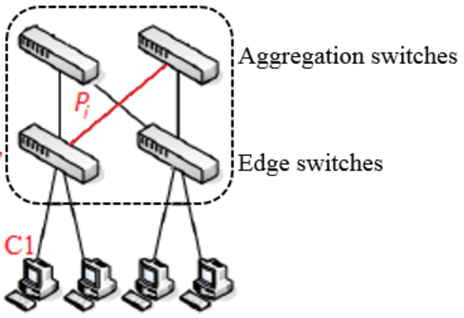


FIGURE 5. Synthetic topology: Composed of 1 pod consists of 2 edges and 2 aggregation switches with single controller.

TABLE 3. Case study experimental details.

Specification	Details
SDN emulator	Mininet v. 2.1
Switch type	Open vSwitch(OVS) v. 2.5.2
Traffic generator	D-ITG
Traffic Type	Randomized TCP/UDP
Network topology	1 Pod of k-pod fat-tree k = 4
Number of flows in pi	10, 20, 30, 40, 50
Number of flows in the switch	100, 200, 300, 400, 500

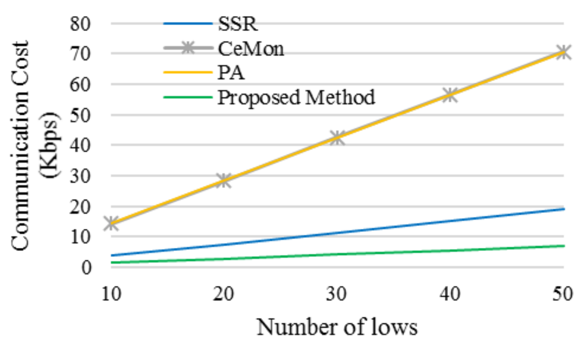
which is directly attached to it. To observe the effectiveness of the proposed solution in single controller, the proposed method is compared versus two native OpenFlow methods (SSR and PA) and CeMon [12]. The flows are generated in a uniformly random fashion using D-ITG [34]. We conduct our experiment in mininet version 2.2.1 [35] with open-Vswitch (OVS) version 2.5.2 [36] to emulate the behavior of an SDN switch. We implemented a prototype of our design as a module in the Floodlight version 1.2 controller [37]. Floodlight version 1.2 supports OpenFlow protocol up to specification 1.4 though it is feasible to extend its features to OpenFlow 1.5 through experimenter. The experiment is conducted on a server with Intel(R) Xeon(R) E3-1270 processor 3.50Ghz and 16GB RAM. Table 3 shows the details of the case study experiment.

B. RESULT

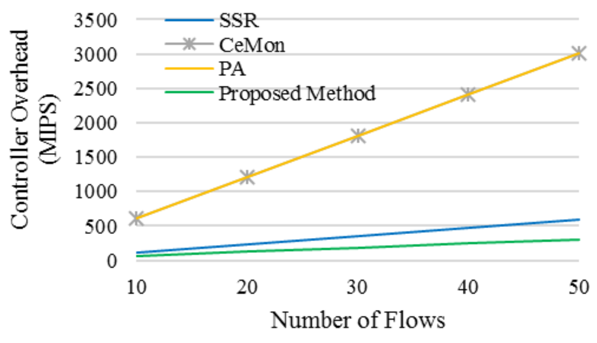
Fig. 6 shows the results of various costs after applying different approaches with different flow number. In section reports the achieved result from our case study.

1) COMMUNICATION COST

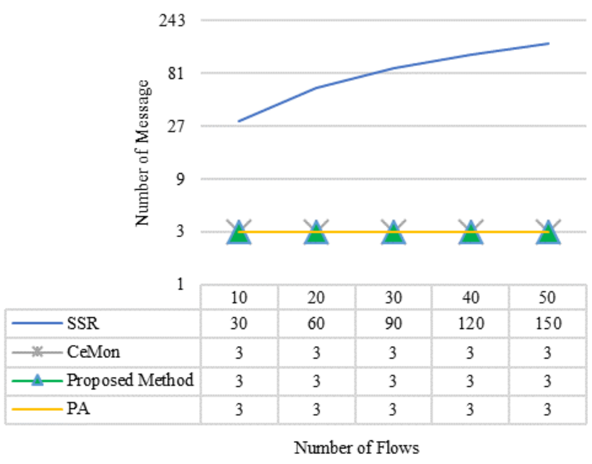
Fig. 6(a) shows the whole costs of communications for all the methods mentioned in section 4.1. The results obtained from our proposed optimal solution indicates a notable reduction of the communication cost. In particular, the proposed solution reported 82% improvement over SSR. The proposed method also saves up to 161% reduction on both PA, and CeMon methods. Basically, pulling all approach can highly alleviate communication cost, however, this cost is highly associated with the total number of flows in the switch which is ten



(a)



(b)



	10	20	30	40	50
SSR	30	60	90	120	150
CeMon	3	3	3	3	3
Proposed Method	3	3	3	3	3
PA	3	3	3	3	3

(c)

FIGURE 6. Various costs after applying different approaches with different flow number. (a) Communication cost. (b) Controller cost (c) Message interaction cost.

times bigger than the actual demanded flows. As it can be seen, the sharp rise in PA and CeMon has a direct correlation to the total number of flows in the switch in every interval. This behavior is arisen for all the methods which apply full or partially ‘‘pulling all’’ strategy in their body. CeMon applies both PA and SSR approach at the same time in which it pulls all flows statistic in the previous intervals (already covered flows) and utilizes a single request for new the flows in the current interval. However, this strategy is highly scenario dependent which is suitable for networks with small rates of new flow arrival or multiple switch selection for pulling.

2) CONTROLLER COST

Fig. 6(b) shows the controller overhead of the proposed method versus the benchmarking methods. The proposed method archives the least controller overhead as it processes only statistics which are associated with the demanding flows (those flows which contribute to the utilization of pi). While, CeMon and PA reported the highest overhead with an explicit increased to the number of flows. This is because these two methods send all the current flows in the switch to be processed in the controller. SSR is distinctively superior to CeMon and PA as a result of sending less number of flows to be processed. This reduction in flows number is due to pulling less flow statistics. Thus, the less flow statistics to pull, the less overhead to be generated. It is observed that the proposed solution significantly saves the overhead by 63% over SSR as a result of reporting the precise flow statistic as demanded.

3) MESSAGING COST

Fig. 6(c) depicts the number of message interaction between the controller and switch. Results obtained from messaging cost reports that the proposed method and CeMon archives the optimal result which is found in the PA approach for message interaction cost. The cost for PA, CeMon and the proposed method remains a constant number and proportionally increases to the number of controller. As there is only one controller in this case study, therefore, for every intervals PA and the proposed method generates three messages (request, reply and synchronized message). CeMon also achieves the optimal number of message interaction as a result of applying ‘‘pulling all approach’’. However, similar to communication cost, the result is highly depended on the scenario and environmental factors such as the number of selected switches to pull or new flows arrival.

Once the solution is presented and evaluated against other approaches and proved to be optimal in a real testbed, a distributed controller problem is defined in the next section then an evolutionary heuristic is presented to evaluate the effectiveness of it.

V. COST EFFECTIVE MULTI-OBJECTIVE CONTROLLER (CEMOC)

Applying multiple controller may result in several unexpected performance degradations such as accuracy and overhead. Each switch can be attached to only one master controller, hereupon assignment[2] of multi-controllers extremely effect on overhead and accuracy. In addition, different deployment of such a scenario highly impacts several factors in the network such as node-to-controller latencies, network availability and performance metrics [38]. Therefore, the controller (place of controller) fetching flow statistic plays a vital role in the accuracy of real-time monitoring as

[2] The assignment and re-assignment of controllers can be referred to as switch selection where the switch(s) may be attached to different controllers in different places.

well as cost, especially in the in-band deployment. In addition to controller assignment, different placement of the coordinator can cause extra cost as well as receiving unsynchronized stats which lead to inaccuracy of the results.

The following described the problem formulations for controller assignment and coordinator placement. Lastly, the proposed multi-controller solution is described based on the optimization of costs and the accuracy of results.

A. PROBLEM DEFINITION

As it was mentioned earlier, the monitoring and routing traffic shares bandwidth along the same path. Therefore, this deployment requires careful planning and precise placement of controllers in the network. Thus, the number of network elements (switches, routers, and cables) can highly effect on the communication costs as well as the accuracy of the result (real-time statistic). Although, certain networking factors such as propagation delay can negatively mutate statistical accuracy. Let donate $h_{\alpha\beta}$ as the number of hops (nodes) from switch v_α to controller c_β and $h_{\beta k}$ as the number of hops from the controller c to the coordinator k. Let w is the cost of communication for pulling flows from a switch. Let donate d as the cost for each controller to be assigned on the switch β and q is the cost of communication from the controllers to the coordinator respectively. Given the propagation delay $\mu_{\alpha\beta}$ for each source-destination pair $v(\alpha, \beta)$, the equations (15-19) describe the integer linear programing of the problem formulation in in-band deployment.

The objective function given by (15) describes the problem formulation of cost which is consists of selecting the most appropriate switch(s) to be pulled (the switch that covers most flows) by w, and the best controller(s) to be assigned given by d_{ck} on the switch(s) for pulling in terms of minimum communication, propagation delay and controller overhead (cost). It also presents the communication between coordinator and all the controllers. Equation (16) refers to the constraint for selecting switch in which at least one switch is selected. Equation (17) explains the constraint for the controller to be assigned on only one switch. Equation (18) forces selecting all the source-destination pairs $v(\alpha, \beta)$ with the least propagation delay. The binary variable used in the formulation are explained as follows:

A binary variable x_β represents whether to pull flow from switch β or not, 1 if it is pulled.

A binary decision variable $y_{ck\beta}$ represents whether a controller ck is assigned on the node β or not, 1 if it is assigned.

$$w_i = \mathrm{l_{rq}} + \mathrm{l_{rp}} + \sum\nolimits_{f_i} \mathrm{l_{sfi}}, \quad f_i \in f,\ f \subseteq F(0 < i < m) \tag{12}$$

$$\begin{aligned} d_{ck} &= \min \sum\nolimits_{ck\in C} \sum\nolimits_{\beta\in V} w_{i\beta}, \\ &\quad \forall ck \in C(0 < i < m), \quad \forall \beta \in V \end{aligned} \tag{13}$$

$$\begin{aligned} co_{cck} &= \min \sum\nolimits_{ck\in C} \left(3\theta + \sum\nolimits_{fi} \lambda_i\right), \quad \forall f_i \in f(0 < i < m), \\ &\quad f \subseteq F, \quad \forall ck \in C \end{aligned} \tag{14}$$

Algorithm 2 *The Eager-greedy approach*

```
Input:   f: sets of flows, $f(0<i<m), \mathrm{f} \subseteq \mathrm{F}$; w: weight of
Output:  pulling set f.
         A: set of groups for pulling $A = \{x \mid x \subseteq F\}$
A ← {}
Covered ← {}
while covered $\neq f$
        j ← calculate (min $w_{if}$)
        if (| $\max_{j\in[0..m]}\{f_j - F\}$ |> 1)
                c ← min $\mu_{\alpha\beta}$ ($\max_{j\in[0..m]}\{f_j - F\}$)
      A ← A U c
        covered ← covered U $f_i$
end while
return A
```

FIGURE 7. The pseudo code of steps involving the selection of switch(s).

Algorithm 3 *Controller Selection*

```
Input:   c: sets of controllers, $c(0<i<n), \mathrm{c} \subseteq \mathrm{C}$; A: set of
         groups for pulling $A = \{x \mid x \subseteq F\}, \forall \alpha, \beta \in \mathrm{V}$
Output:  B: sets of controllers, $c(0<i<n), \mathrm{c} \subseteq \mathrm{C}$
B, n ← {}
foreach pulling set in A
        j ← calculate (min $co_{cA}$ and $h_{A\alpha\beta}$)
        if ($|j_A| > 1$)
                n ← calculate (min $j_{A\beta k}$)
        B ← B U $n_j$
      A ← A U c
end foreach
return A
```

FIGURE 8. The steps involving the selection of controllers.

$$\begin{aligned} &\min \sum\nolimits_{\beta\in v} \sum\nolimits_{i\in f} w_{\beta i} x_\beta \\ &\qquad + \sum\nolimits_{\beta\in v} d_{ck\beta} y_{ck\beta} + \sum\nolimits_{\alpha\in|V|} q_\alpha, \\ &\mathrm{x}_{\beta\mathrm{i}}, y_\alpha, z_\alpha \in \{0, 1\}, \forall ck \in C, \forall \alpha, \beta \in V \end{aligned} \tag{15}$$

Subject to:

$$\sum\nolimits_{\beta\in|V|} \sum\nolimits_{\mathrm{i}\in\mathrm{f}} w_{\beta i} x_\beta > 1, \quad \forall \beta \in \mathrm{V}, \mathrm{q} \in \{0, 1\} \tag{16}$$

$$\sum\nolimits_{\beta\in|V|} \mathrm{y}_{\mathrm{ck}\beta} < 1, \quad \forall \beta \in \mathrm{V}, \mathrm{y} \in \{0, 1\} \tag{17}$$

$$\sum\nolimits_{h_{\alpha_{\beta\in|V|}}} \mu_{\alpha\beta} < \min\{v(\alpha, \beta)\}, \quad \forall \alpha, \beta \in \mathrm{V} \tag{18}$$

The minimization problem defined above is a weighted set cover problem which is proven to be NP-hard and requires heuristics to approximate the performance. Note that, propagation delay is computed for a link between two switches as it is determined by LLDP send/receive times minus the delay between both switches and the controller. However, this delay does not cause a significant effect in datacenters as it is on the order of microseconds.

B. SOLUTION

The easiest way to solve the problem of weighted set cover is to apply the brute force search algorithm known as exhaustive search in which it enumerates all possible candidates for the

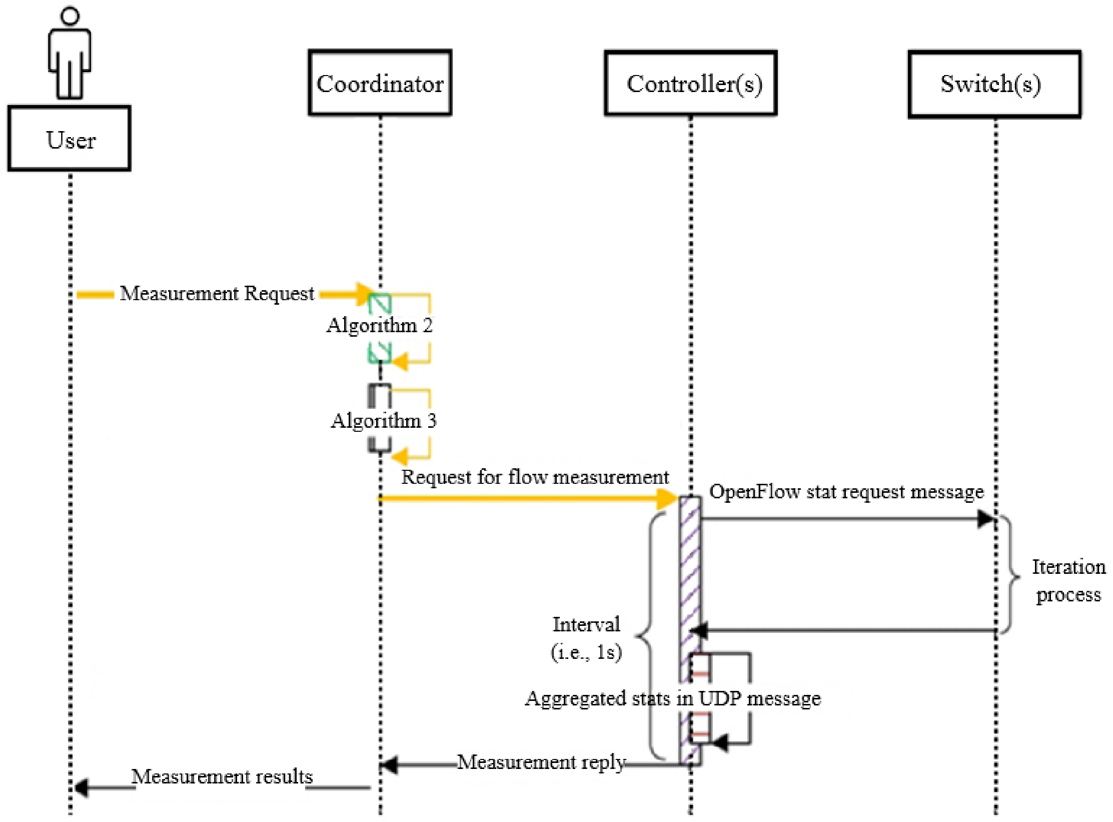


FIGURE 9. The flow process of The CeMOC.

solution and checks the correctness of the solution. Although, the brute search algorithm is simple and always find the solution, its time complexity is $O(2m+n)$ which is exponential to the number of flows and switches m and n respectively which is not scalable for datacenters nor ISPs. Hence, for large scale networks, approximation technique is required.

To solve the problem above, we apply an Eager-greedy algorithm which is an approximation technique adopted from [39]. This algorithm implements a priority queue to alleviate the time complexity to *O(m log n)*. We modified the algorithm to select the most cost-effective switch(s) that cover all demanding flows based on their given weight (w_i). Fig. 7 shows the steps involving the selection of switch(s) in pseudo code. In each iteration, it calculates the minimum associated weight (shortest path) for all demanding flows in step 4. It then identifies the sets with the largest number of uncovered items in step 6 and put it in the output as a group in step 7. If the algorithm finds more than one sub-covered set (step 5), it selects the subset with the least propagation delay in step 6.

We also propose an algorithm that assigned a controller to a switch for the pulling purpose. Note that, only one controller can be assigned to a switch in a master mode. Fig. 8 illustrates the steps involving the selection of controllers in pseudo code. The main loop iterates for $O(n)$ time where $n = |A|$ which is the number of pulling set(s). It then calculates the nearest controller with the least CPU load to the pulling set (switch(s)) in step 3 with the time complexity of $O(m^2)$ where $m = |C|$ which is the number of controllers. In the case of finding more than 1 controller for a set, it selects the nearest one to the coordinator place. Fig. 9 illustrates the entire flow process of the CEMoC. As it is sketched in the Fig. 9, algorithm 2 and 3 run only once upon a measurement request to the coordinator. Then coordinator signals the selected controllers and pulling set(switches) to be pulled. The life line of controller(s) continues for the number of iteration. Therefore, every controller sends stat request and receives the stat reply until the signaling of user for termination of the process.

VI. EVALUATION

We evaluated the performance of CEMoC with extensive network emulations from various perspectives in different scenarios. In this section, we present experimental setup, result, and analysis of the evaluation.

A. EVALUATION SETUP

To emulate the network topology and flow generation, we used a single Amazon EC2 m4.4.xlarge instance with operation system Ubuntu 16.04 Server. We used Mininet version 2.2.1 by which it uses a Linux container to emulate hosts and OpenvSwitch (OVS), which allows the entire network to be emulated in a single computer. We implemented Open

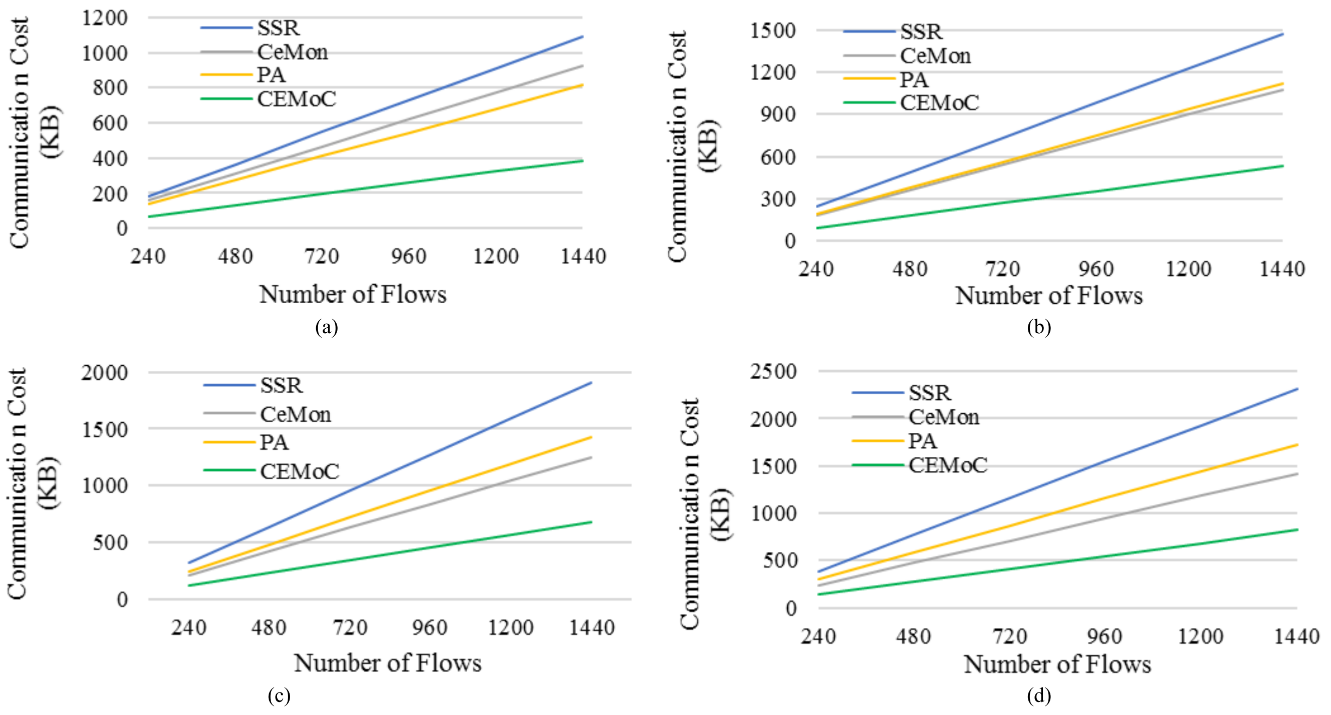


FIGURE 10. **Total communication cost with different number of controllers. (a) Communication cost in 4 controllers. (b) Communication cost in 3 controllers. (c) Communication cost in 2 controllers. (d) Communication cost in 1 controller.**

vSwitch version 2.5.5 for emulating routers/switches behavior. All the links in our implementation are set to 30Mbps due to the limited capacity in a single machine. We implement 4 prototypes of CEMoC as a module of the Floodlight controller and placed them in a first host attached to each pod. We also place the coordinator as a desktop application in the first host of first pod. We set all the connection as in-band network deployment. The topology used in our evaluation is 4-port fat tree replicated from [40]. We generate flows by D-ITG [34] traffic generator which has been proven to perform reliable and scalable than other traffic generator.

Benchmark and Workloads: The objective of this evaluation was to measure all the UDP flows traffic with a specific destination port number (i.e. 3660) in the datacenter network. Every host in the network starts randomly sending UPD traffic with the dest port number above. To ensure the correctness of results and the pattern (behavior) of outcome, we iterated the evaluation 10 times with 6 different flow number. We repeat all of iterations for different controller number. Two different traffic scenarios Constant and Variable Bit Rate (CBR and VBR) are used to evaluate different traffic workload and shape to show the effectiveness of CEMoC on costs and accuracy over a realistic workload pattern. VBR follows the traffic pattern introduced in [31] (shape and scale) for all the generated flows in the network.

CBR: In this model, all the hosts generate UDP flows in the constant increased rate and sizes. All hosts generate a new flow every second and send it over all other hosts. In the first cycle, each host sends two UPD flows (with dest port 3660 and a random src port number) to all other hosts. The number of flows is increased to twice in the next cycle and so forth to the last iteration. Thus, there are 480 and 2880 flows in the first and last iteration respectively.

VBR: Applying D-ITG Pareto distribution for the inter-departure of times of the packet. We used $\varphi = 1.75$ as the shape parameter for all the flows and a random scale parameter γ from the range 0.5ms to 1ms.

Performance Evaluation: In order to evaluate the effectiveness of CEMoC, we performed the evaluation and analyzed the given result with different number of controllers (from 1 to 4). We then show the result of CEMoC in various performance metrics, such as communication overhead, message interaction, controller overhead, and the accuracy of the result. Moreover, we evaluated the measurement error caused by different network delays. Table 4 shows the specification used in experiments.

B. EVALUATION RESULT AND ANALYSIS

In order to fully understand different costs caused by pulling the switches, we iterated the mininet emulation scenario for 10 times with a wide range of flow numbers. We conducted our emulations to show the performance of CeMOC against SSR, PA and MCPS proposed in CeMon [12] in multiple controller scenario with different controller number. We also explain our observation of accuracy and error rates in multiple controller scenario. We run the emulation for 4 times with different controller number and iterated them with different network delays (such as 0, 5, 10, 25, 100ms).

TABLE 4. Specification of experiment.

Spec	Type
Host vCPU	16
Host Memory	64Gb
Host OS	Ubuntu 16.04 Server
Mininet version	2.2.0
OVS vesion	2.5.2
Floodlight version	1.2
Traffic Generator	D-ITG version 2.8.1
Traffic Pattern	CBR VBR (φ = *1.75*, γ = *0.5 to 1*)
Network Topology	Fat-tree k-pod (k=4)
Number of Host	16
Number of Flow	240, 480, 720, 960, 1200, 1440

Number of Switches	Edge = 8
	Aggregation = 8
	Core = 4

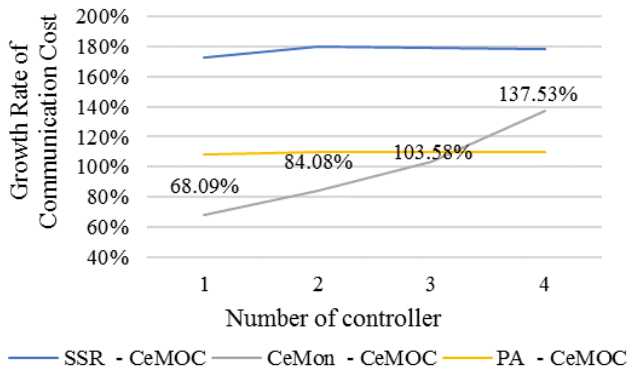


FIGURE 11. Average growth rate of communication cost in different number of controllers.

1) COMMUNICATION COST

Fig. 10 shows the communication cost with different number of controller. Obviously, the CeMOC is constantly superior to all other methods with different number of flows. We observed a linear increase in all the methods as the number of flows grow. CeMOC reported the lowest communication cost by maximum 386Kbps when flow number is 1440. We observed that the CeMOC saves up to 97, and 138 in comparison with PA, and SSR, respectively. However, findings from sub-Fig. a, b, c, and d in the Fig. 10 demonstrated that the increment rate of communication cost varies with different number of controllers. Fig. 11 explains the average growth ratio of communication cost for each method over CeMOC where SSR, and PA show a constant growth rate whereas, CeMon demonstrates the least increment rate with 68% in single controller and biggest change when there are 4 controllers in the network. This is because CeMon selects both core and edge switches, consequently, it applies pulling all approach and single flow request respectively. In addition, the number of paths from the controller to switches is increased as more switches are attached indirectly through in-band data paths. Thus, more switches to pull, results in more flow statistics to communicate. It also indicates that there is a slight but steady growth of SSR, and PA over CeMoC when the number of controller increases. However, CeMon shows that it decreases communication cost by the decrease of controller number.

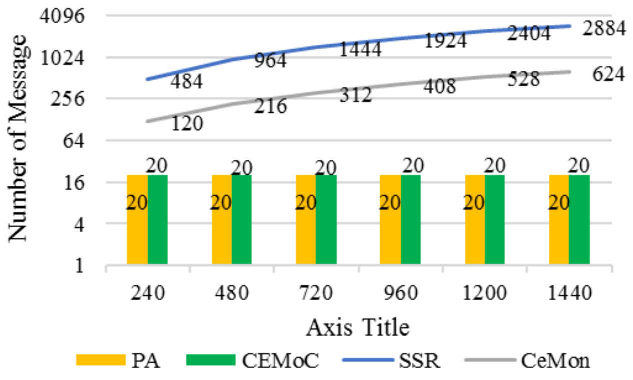


FIGURE 12. Message interaction in 4 controllers.

2) MESSAGE INTERACTION

Fig. 12 shows message interaction with 4 controllers, where CeMOC and PA achieved the most efficient number of message interaction in all of the iterations. However, PA sacrifices other costs (such as communication and controller overhead) at the expense of low message interaction. This is because PA aggregates the whole flow stats in the switch in only one message. The efficient number of message interaction is a constant number which can be found in the methods, which apply pulling all flows. This strategy saves excessive sending and receiving messages from controllers and switches. CeMon stands in the third place as it can reduce almost half of the iterations using pulling all from core switches. However, we observe that the reply messages are split into two messages when the number of flows is more than 453 in core switches. SSR demonstrates the highest number of interactions as a result of pulling switches for every individual flow. Totally, CeMOC could achieve a reduction in message interaction up to 175%, and 195% over CeMon, and SSR respectively. Table 5 shows the result of message interaction with different number of controllers. It is observed that all the methods achieve a slight decrease when the number of controllers is reduced.

3) CONTROLLER OVERHEAD

Fig. 13 explains the total overhead of 4 controllers with different flow numbers. It is observed that the overhead of all the methods is linearly increased at the expense of flow growth. Thus, the more flows are counted, the more overhead is impressed. Unlike PA and CeMon which reports all the flows' stats for calculation, CeMOC reports only required statistic which contributes to the utilization of UDP flows. As a result, there are averages of 126% of reduction in controller overhead by CeMOC over PA. However, SSR reports

TABLE 5. Message interaction cost with different number of controller.

Benchmarks / Flows	240	480	720	960	1200	1440
	Number of Message Interaction					
	3 Controllers					
SSR	483	963	1443	1923	2403	2883
CeMon	118	214	310	406	526	622
PA	19	19	19	19	19	19
CEMoC	19	19	19	19	19	19
	2 Controllers					
SSR	482	962	1442	1922	2402	2882
CeMon	116	212	308	404	524	620
PA	18	18	18	18	18	18
CEMoC	18	18	18	18	18	18
	1 Controller					
SSR	481	961	1441	1921	2401	2881
CeMon	114	210	306	402	522	618
PA	17	17	17	17	17	17
CEMoC	17	17	17	17	17	17

TABLE 6. Controller overhead with different number of controller.

Benchmarks / Flows	240	480	720	960	1200	1440
	Controller cost with different flow number					
	3 Controllers					
SSR	2403	4803	7203	9603	12003	14403
CeMon	3286	6550	9814	13078	16342	19606
PA	2915	5795	8675	11555	14435	17315
CEMoC	1475	2915	4355	5795	7235	8675
	2 Controllers					
SSR	2402	4802	7202	9602	12002	14402
CeMon	3284	6548	9812	13076	16340	19604
PA	2914	5794	8674	11554	14434	17314
CEMoC	1474	2914	4354	5794	7234	8674
	1 Controller					
SSR	2401	4801	7201	9601	12001	14401
CeMon	3282	6546	9810	13074	16338	19602
PA	2913	5793	8673	11553	14433	17313
CEMoC	1473	2913	4353	5793	7233	8673

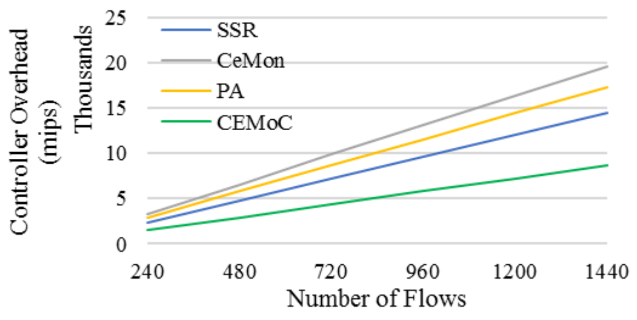


FIGURE 13. Controller overhead in 4 controller scenario.

stats in the same way as CeMOC. The difference between SSR over CeMOC is the generation of files and reading them with regard to the flow number as every flow is placed in one file. Nevertheless, there is only one file to read as CeMOC aggregates all the required stats in only one file. CeMOC saves up to 65% of controller overhead over SSR. We also observed that the total controller overhead is reduced by less than 0.01% on average when the number of controller is decreased. Table 6 reports controller overhead with different number of controller.

4) ACCURACY

Unlike statistical estimation model or sampling methods which is used in traditional networks, the accuracy in our work mainly corresponds with the time. Basically, due to network latency and sequential creation of messages in the controller, synchronizing pulling requests is infeasible for all the switches in a network. Also, the exact moment of reading flow counters in the switches is unknown [31]. As a consequence, estimating flow utilization may be limited by a negligible error rate. This problem is also referred to as ''Accuracy limitation'' due to lack of synchronization which can be more sophisticated when dealing with in-band deployment where statistic request and result traverse through network's data plane paths. We conveyed a 360 seconds experiment with the VBR traffic pattern to highlight the observed error and the impact of different controller number on accuracy.

Fig. 14 shows the actual measured flow utilization captured by Wireshark, CeMOC and the relative error with 4 controllers and no extra delay (Ideal case). It can be observed that the flows' utilization captured by CeMOC is very closed to the actual one. In fact, CeMOC reports the maximum absolute error and standard deviation of 9.49% and 1.98% respectively. We introduced artificial delays in the network to understand the impact of different number of controller in accuracy. Table 7 illustrates the relation between Error ratio on different controller number and delays. It can be observed that the error ratio increases at the expense of increasing delay in different controller number. Thus, the more delay is introduced, the bigger error ratio is exposed. However, from table 7 it is concluded that number of controller has a direct relationship with the error ratio by which the error ratio for all the delays are decreased when the number of controller is increased and vice versa.

In order to construct a full traffic matrix (collecting all flows' stats from all controllers) all controllers send the counted flow bytes (stats) and aggregate them into a UDP file and then send them to the coordinator. Recall that all the transmissions in out-of-band deployment take place through the data plan network links. Table 8 shows the maximum transferring delay of final UDP packet from each controller to the coordinator. Therefore, in the worst case a packet in a fat-tree topology may go through 5 switches and links each of which may impose different delay to the packet until it reaches the destination (coordinator). Therefore, in the worst case, CeMOC is able to record flows' utilization without overlapping in the next intervals. However, we proactively

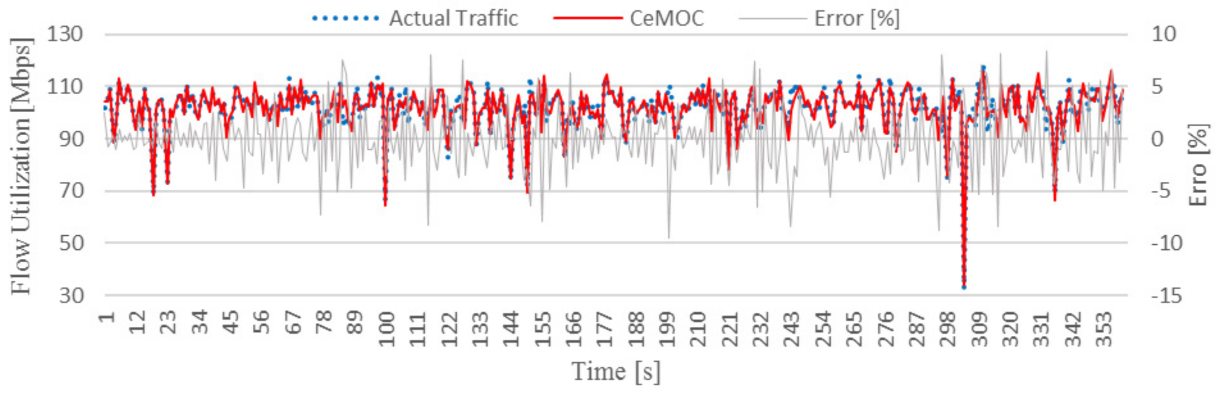


FIGURE 14. Actual measured flow utilization captured by Wireshark, CeMOC and the relative error.

TABLE 7. The relation between error ratio on different controller number and delays.

Error \ Delay	0	5	10	15	100
	Relative Error with different delay [%]				
	Relative Error with 4 Controllers				
Average	0.07	0.08	0.09	0.11	0.20
STD	3.03	3.01	3.01	3.02	3.72
Max Absolute Error	9.48	9.46	9.43	9.36	9.00
	Relative Error with 3 Controllers				
Average	0.07	0.08	0.09	0.11	0.20
STD	3.03	3.01	3.01	3.02	3.72
Max Absolute Error	9.48	9.44	10.10	16.20	46.40
	Relative Error with 2 Controllers				
Average	0.07	0.08	0.09	0.12	0.26
STD	3.02	3.01	3.00	3.06	4.47
Max Absolute Error	9.47	9.43	11.70	19.50	58.60
	Relative Error with 1 Controllers				
Average	0.08	0.0%	0.11	0.15	0.37
STD	3.02	3.00	3.02	3.22	6.29
Max Absolute Error	9.46	10.00	13.20	22.80	70.80

TABLE 8. Maximum transferring delay of final UDP packet from each controller to the coordinator.

Network Delay [ms]	0	5	10	15	100
Controller to Coordinator [ms]	14.2	39.7	68.9	159.4	606.6

installed the corresponding flow entry to the switches to transfer the UPD packet from the controllers to the coordinator. We set a static flow entry (from controllers to coordinator) in every switch in the network to eliminate the aforementioned delay above.

VII. CONCLUSION AND FUTURE WORK

This paper introduced CeMOC, a cost-efficient flow measurement system in distributed (multiple) controller deployment for datacenter networks. We designed a two-layered architecture by which the SDN controller(s) pull the switches to collect flow statistics in the first layer, and transfer aggregated flow stats to a coordinator in the second layer. We adopted group table feature of OpenFlow 1.3 with active measurement to pull a set of desired flow stat in one reply. A case study with a synthetic topology was conducted to show the feasibility and reliability of the proposed design in out-of-band network deployment. CeMOC was prototyped as a standard SDN northbound RESTful API in floodlight controller which is able to perform almost all aspects of monitoring system. We introduced a selection algorithm to optimally select a set of switches for pulling purpose. We conducted an extensive experiment in in-band network deployment whose findings indicated that CeMOC can capture near real-time traffic with a significant reduction in communication cost, message interaction, and controller overhead. We also analyzed the result of our experiment for the accuracy of CeMOC measurement task with different controller's number and delays. Finding indicates that the controller number significantly effects on the communication cost and accuracy, more controllers less costs with more accurate result. We applied mathematical evaluation on controller cost using CPU mips calculation, However, we believe empirical evaluation needs to be taken into account to measure an accurate controller cost for memory footprint in each controller. Furthermore, the paper's formulation was mainly proposed for the topologies with multiple stages of switches (such as fat-tree). However, it is generalizable for other datacenter types where the topology applied is Clos like topology. We will study more network topology types and apply our proposed design on other types of network to observe the effectiveness of our design and its performance.

REFERENCES

[1] C.-W. Chang, G. Huang, B. Lin, and C.-N. Chuah, "LEISURE: Load-balanced network-wide traffic measurement and monitor placement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 1059–1070, Apr. 2015.
[2] B. Claise, *Cisco Systems NetFlow Services Export Version 9*, document RFC 3954, Internet Engineering Task Force, 2004.
[3] P. Phaal and M. Lavine. (2004). *sFlow Version 5*, InMon Corp. [Online]. Available: http://sflow.org/sflow_version_5.txt

[4] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in presented as the 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI), 2013, pp. 29–42.

[5] G. R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran, "Reformulating the monitor placement problem: Optimal network-wide sampling," in *Proc. ACM CoNEXT Conf.*, Mar. 2006, pp. 1725–1731.

[6] H. Xu, Z. Yu, C. Qian, X.-Y. Li, and L. Huang, "Minimizing flow statistics collection cost using wildcard-based requests in SDNs," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3587–3601, Dec. 2017.

[7] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 7–12, Oct. 2013.

[8] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: A survey," *Comput. Commun.*, vol. 67, pp. 1–10, Aug. 2015.

[9] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for OpenFlow networks," in *Proc. Int. Conf. Passive Active Netw. Meas.*, 2010, pp. 201–210.

[10] 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. Operat. Manage. Symp. (NOMS)*, May 2014, pp. 1–9.

[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 Active Netw. Meas.*, 2013, pp. 31–41.

[12] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "CeMon: A cost-effective flow monitoring system in software defined networks," *Comput. Netw.*, vol. 92, pp. 101–115, Dec. 2015.

[13] 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.

[14] A. C. Myers, "JFlow: Practical mostly-static information flow control," in *Proc. 26th ACM SIGPLAN-SIGACT Symp. Principles Programm. Lang.*, 1999, pp. 228–241.

[15] V. Mohan, Y. J. Reddy, and K. Kalpana, "Active and passive network measurements: A survey," *Int. J. Comput. Sci. Inf. Technol.*, vol. 2, no. 4, pp. 1372–1385, 2011.

[16] S. Sezer *et al.*, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013.

[17] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "CPI^2: CPU performance isolation for shared compute clusters," in *Proc. 8th ACM Eur. Conf. Comput. Syst.*, 2013, pp. 379–391.

[18] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. OSDI*, Vancouver, BC, Canada, 2010, pp. 1–6.

[19] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," presented at the Internet Netw. Manage. Conf. Res. Enterprise Netw., San Jose, CA, USA, 2010.

[20] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," presented at the 3rd Workshop Hot Topics Softw. Defined Netw., Chicago, IL, USA, 2014.

[21] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw.*, Jun. 2014, pp. 1–6.

[22] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," presented at the 1st Workshop Hot Topics Softw. Defined Netw., Helsinki, Finland, 2012.

[23] B. Pfaff, B. Lantz, and B. Heller, "OpenFlow switch specification, version 1.3.0," Open Netw. Found., Tech. Rep., 2012.

[24] B. Pfaff, B. Lantz, and B. Heller, "OpenFlow switch specification, version 1.1.0," Open Netw. Found., Tech. Rep., 2011.

[25] R. Ahmed and R. Boutaba, "Design considerations for managing wide area software defined networks," *IEEE Commun. Mag.*, vol. 52, no. 7, pp. 116–123, Jul. 2014.

[26] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Fast failure recovery for in-band OpenFlow networks," in *Proc. 9th Int. Conf. Design Reliable Commun. Netw. (DRCN)*, 2013, pp. 52–59.

[27] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Commun. Surveys Tuts.*, vol. 9, no. 3, pp. 44–57, 3rd Quart., 2007.

[28] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee, "DevoFlow: Cost-effective flow management for high performance enterprise networks," presented at the 9th ACM SIGCOMM Workshop Hot Topics Netw., Monterey, CA, USA, 2010.

[29] D. Sünnen, "Performance evaluation of openFlow switches," Ph.D dissertation, Dept. Inf. Technol. Elect. Eng., Swiss Federal Inst. Technol. Zürich, Switzerland, 2011.

[30] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design MIPS Edition: The Hardware/Software Interface*, 5th ed. New York, NY, USA: Elsevier, 2014, p. 800.

[31] P. Megyesi, A. Botta, G. Aceto, A. Pescapé, and S. Molnár, "Challenges and solution for measuring available bandwidth in software defined networks," *Comput. Commun.*, vol. 99, pp. 48–61, Feb. 2017.

[32] K. Yi and Y. H. Ding, "32-bit RISC CPU based on MIPS instruction fetch module design," in *Proc. Int. Joint Conf. Artif. Intell.*, 2009, pp. 754–760.

[33] R. Izard. (2016). *Fast-Failover OpenFlow Groups*. [Online]. Available: https://floodlight.atlassian.net/wiki/display/floodlightcontroller/How+to+Work+with+Fast-Failover+OpenFlow+Groups

[34] A. Botta, A. Dainotti, and A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios," *Comput. Netw.*, vol. 56, no. 15, pp. 3531–3547, Oct. 2012.

[35] Mininet. (2015). *Mininet Version 2.2.1 ed.* [Online]. Available: http://mininet.org/overview/

[36] OpenvSwitch. (Feb. 17, 2017). *Open vSwitch Version 2.5.5*. [Online]. Available: http://openvswitch.org/releases/NEWS-2.5.2

[37] BigSwitchNetworks. (2016). *Floodlight v1.2*. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/24805419/Floodlight+v1.2

[38] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Comput. Netw.*, vol. 112, pp. 279–293, Jan. 2017.

[39] C. L. Lim, A. Moffat, and A. Wirth, "Lazy and eager approaches for the set cover problem," presented at the 37th Austral. Comput. Sci. Conf., vol. 147, Auckland, New Zealand, 2014.

[40] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

HAMID TAHAEI received the M.Sc. degree in computer science from the University of Technology Malaysia, Johor, Malaysia, in 2013. He is currently pursuing the Ph.D. degree in the field of network security from the University of Malaya, Malaysia. His research interests include software defined networks, traffic engineering, IoT, and cloud computing security.

ROSLI BIN SALLEH received the B.S. degree in computer science from the University of Malaya, Malaysia, in 1994, and the M.Sc. and Ph.D. degrees from the University of Salford, U.K., in 1997 and 2001, respectively. Since 2001, he has been a Lecturer with the Department of Computer System and Technology, Faculty of Computer Science and Information Technology, University of Malaya. He was appointed as a Senior Lecturer in 2007 and an Associate Professor in 2013. His research interests include SDN, mobile IPv6 handover and security, botnet, and wireless sensor networks. His awards and honors include the Frew Fellowship from the Australian Academy of Science, the I. I. Rabi Prize (APS), the European Frequency and Time Forum Award, the Carl Zeiss Research Award, the William F. Meggers Award, and the Adolph Lomb Medal (OSA).

MOHD FAIZAL AB RAZAK received the master's degree in computer science (networking) from University Malaysia Pahang, Malaysia. He is currently pursuing the Ph.D. degree from the University of Malaya, Malaysia. His area of research includes mobile computing and mobile security.

KWANGMAN KO received the B.Sc. degree from Wonkwang University in 1991, and the M.Sc. and Ph.D. degrees in computer engineering from Dongguk University, South Korea, in 1993 and 1998, respectively. He is currently a Professor with the Department of Computer Engineering, School of Computer and Information Engineering, Sangji University, South Korea. His research interests include the re-targetable tool suite for embedded systems, virtual machines, and architecture description language.

NOR BADRUL ANUAR received the master's degree in computer science from the University of Malaya, Malaysia, in 2003, and the Ph.D. degree in information security from the Centre for Security, Communications and Network Research, Plymouth University, U.K., in 2012. He is currently an Associate Professor with the Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur. He has published a number of conference and journal papers locally and internationally. His research interests include information security (i.e., intrusion detection systems), data sciences, artificial intelligence, and library information systems.