A Sketch-based Network Traffic Analysis Implementation on Commodity OpenFlow Switches

Zong-Cheng Liou†, Chung-Hsiang Cheng†, Theophilus Wellem*, Ku-Yeh Shih†, and Yu-Kuen Lai†
†Department of Electrical Engineering, *Department of Electronic Engineering
Chung Yuan Christian University, Chung-li 32023, Taiwan
Email:{g10379006, g10578017, g10202604, g10678604, ylai}@cycu.edu.tw

***Abstract*—The paradigm of Software-defined Networking (SDN) provides excellent flexibility where the traffic monitoring applications can be implemented in the SDN controller. The objective of this work is to develop a software implementation of a Sketch-based network traffic analysis system on commodity OpenFlow switches. A sketch data structure is implemented inside the Open vSwitch as the extra table to collect flow statistics. The OpenFlow protocol is extended to support sketch request and reply messages such that the controller can collect the sketch data structure from the switches. The implementation of heavy-change detection is provided to showcase the flexibility of this implementation. Performance evaluations are conducted with real-world network traffic traces. The system is capable of finding flows with significant volume changes above a predefined threshold.**

***Index Terms*—Software-defined networking; Change detection; OpenFlow switch.**

I. Introduction

Software-defined networking (SDN) is an emerging network architecture [1] that decouples the network control plane from the data plane. The control plane is typically located in a commodity server, known as the SDN controller. The controller communicates with the network devices in the data plane using an open interface such as OpenFlow protocol. The SDN paradigm has enabled a new era of software-defined traffic monitoring that can provide flexible flow-based traffic measurement [2]–[4]. Applications (*e.g.,* an anomaly detector) running in the SDN controller can dynamically modify the rules of flows to be monitored based on the network status and policy set by the administrator. Furthermore, software-defined traffic monitoring has the potential to enable concurrent and dynamically instantiated network measurement tasks [5].

In a typical OpenFlow-based SDN architecture, the operation of a monitoring application relies on the support of flow entries in the flow tables. The application obtains the flow statistic counters from switches and processes the data using sophisticated algorithms and statistical methods. This type of applications, utilizing the flow statistics gathered from the switch's flow table, is widely used in the deployment of SDN-based network traffic monitoring [6], [7].

However, the capability of traffic monitoring, relying on the flow entries in the flow table of the commodity hardware OpenFlow switches, is limited by the TCAM size. Typically, wildcard rules are installed in the TCAM along with the forwarding rules by the controller for traffic monitoring purpose [8]. Flow monitoring rules, often defined at different granularity [9], may also interfere with forwarding rules already presented in the switches. Moreover, the main purpose of the TCAM in OpenFlow switch is to store the rules for forwarding traffic, not for monitoring purpose. Therefore, the monitoring capabilities are neither flexible nor scalable.

One approach to resolve the issues is to decouple the monitoring task from the use of the flow table. This can be achieved by adding an extra table or data structure in the data plane for monitoring and measurement tasks. Examples of this approach can be found in the OpenSketch [2] and UnivMon [10] that utilize sketch data structure in the data plane. Another work [11] uses Bloom filters to select flows to be measured and determine the action to be executed on matched flows. In [12], the Count-Min sketch [13] is put inside an FPGA-based OpenFlow switch for heavy hitter detection.

In this paper, we first present the *typical* implementation of sketch-based traffic change detection by collecting the flow statistics periodically from the commodity hardware OpenFlow switches. Then, the *In-switch Sketch* implementation in an Open vSwitch [14] are discussed. The sketch is implemented inside the switch as the extra table to collect flow statistics. A controller can retrieve the sketch data structure by using the OpenFlow protocol. We have modified the OpenFlow protocol to support sketch *request* and *reply* messages such that the controller can collect the sketch data structure from the switches. After receiving the sketch, the monitoring application can use the sketch for traffic change detection.

The main contributions of this work are as follows.

- We design and implement a sketch-based traffic change detection system in SDN using commodity hardware OpenFlow switches. Furthermore, analysis on factors that influence the detection accuracy is presented.
- The implementation of sketch data structure in an Open vSwitch supporting traffic monitoring is developed. OpenFlow protocol is extended with messages to handle the sketch data processing between switches and the controller.

The remainder of this paper is organized as follows. OpenFlow, SDN traffic monitoring, and sketch-based change detection are introduced in Section II. Section III describes the typical and the in-switch sketch system implementation. The system evaluation and discussion of the experiment results

are presented in Section IV. Finally, Section V concludes this paper with future works provided.

II. Background and Related Work

A. OpenFlow and SDN traffic monitoring

The typical OpenFlow approach for traffic monitoring is to use the flow statistics recorded in the switch's flow table. The mechanism to obtain the flow statistics can be either active or passive. In active mode, the controller sends the request message to the switch and the switch responds by sending the flow statistics reply message back to the controller. In passive mode, the controller waits for the messages sent by the switch. When flow entries with special flag have expired (timeout), the switch notifies the controller on removing the flow entries. Then, the controller can obtain the statistics of the flow entries to be removed. The controller is responsible to install flow entries with special flag in the switch beforehand.

Scalability challenges for network monitoring applications in SDN are discussed [4] [15]. These challenges, consisting of limited TCAM size and limited CPU capacity processing the statistics inside the switch [16] affect the quality of measurement with inconsistent and inaccurate results.

Based on the OpenFlow specification, presenting the OpenFlow logical switch requirements and the OpenFlow protocol, the switch has stateless data plane and uses match-action abstraction. The drawback of OpenFlow for network monitoring is due to its stateless data plane that prevents implementation of data plane algorithms [15]. Applications must rely on the controller for stateful operations. Therefore, several researches [17], [18] on stateful SDN data plane and monitoring are proposed. Stateful monitoring solutions such as OpenSketch and UnivMon proposed the usage of sketch data structure as the monitoring primitive on the data plane. Therefore, extra memory space is dedicated for the monitoring tasks, specifically for data collecting and pre-processing.

Research works such as UMON [19] and SDN-Mon [20] are proposed to decouple the operations of monitoring from forwarding without mutual interference. In UMON, a separate monitoring flow table is used to store the monitoring rules. SDN-Mon is an SDN-based monitoring framework that decouples monitoring from forwarding table. It allows the controller to define a set of monitoring match fields based on the requirements. The monitoring fields are stored in another table and a Bloom filter is used to determine whether the monitoring table should be checked and updated during processing the network traffic.

The work presented in this paper follows the forwarding-monitoring decouple paradigm and sketch is used as the monitoring primitive.

B. Heavy change detection

The goal of heavy change detection is to identify flows with significant changes in size (packet counts) or volume (byte counts) above a predefined threshold of the total traffic over several consecutive observation intervals. These flows are also known as *heavy changers*. Let $A_t = a_1, a_2, \ldots$ be the input packet stream in observation interval t. Each packet a_i is associated with a key k and an updated value v. Following the algorithm in [21], the source IP address and packet length are used as the key and the updated value, respectively. The algorithm uses *k*-ary sketch, which is an array of counters S with H rows and K columns. H hash functions are used, where each row is associated with a hash function. Let $S_o(t)$ is the sketch that accumulate the byte counts in t and $S_f(t)$ is the forecast sketch computed using a forecasting model (e.g., moving average). The forecast error sketch is then computed as $S_e(t) = S_o(t) - S_f(t)$. A threshold alarm T_A is chosen based on the estimated second moment of the forecast error sketch and a threshold parameter T determined by application ($T_A = T \times \sqrt{EstF_2(S_e(t))}$). The change detection task is to find all keys (source IP addresses) whose volume, queried from $S_e(t)$, is above T_A.

III. System Implementation

A. Typical Implementation

The change detection application is implemented and executed in the controller following the typical OpenFlow approach for traffic monitoring[1]. Two configurations of *standard* and *parallel* are used for the traffic change detection to utilize all of the hardware resources (flow tables, memories) available on the switches. In the standard configuration, only one switch is used and the controller sends request to the switch periodically to obtain the flow statistics. After receiving the flow table entries, the application updates its sketch data structure, computes the threshold, and executes the change detection algorithm. The bottleneck in this configuration is in the CPU capacity and the TCAM size of the switch. If the traffic is too large, the switch is not able to process the traffic and produces inaccurate detection result. To solve this problem, parallel configuration is used, in which multiple switches work together. This configuration improves the detection accuracy because more flow statistics can be utilized by the change detection process. The parallel configuration maximizes the hardware resource usage in multiple switches. In this configuration, the traffic is distributed to all of the switches such that each switch only needs to process a part of the traffic, thereby reducing the switch's CPU load. The distribution mechanism is done using wildcard rules according to network prefix. With wildcard rules, match can be done on larger set of fields. The system can insert predefined wildcard rules and actions such that the traffic will be directed to each switch in parallel configuration. Commodity hardware OpenFlow switch and Floodlight OpenFlow controller [22] are used in the implementation.

The change detection system, consisting of *statistics requester* module, *change detection* module, *record* module, and *accuracy calculation* module is shown in Fig. 2. The monitoring process starts when the connected switches are initialized and allocated according to the configuration selected

[1]We refer this implementation as the "typical implementation" throughout this paper.

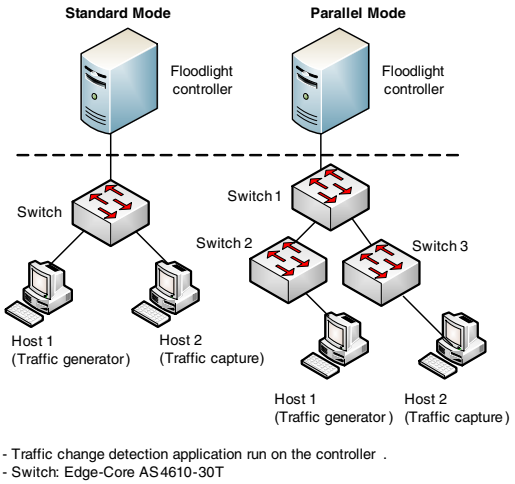

Fig. 1. Configurations of the traffic change detection system.

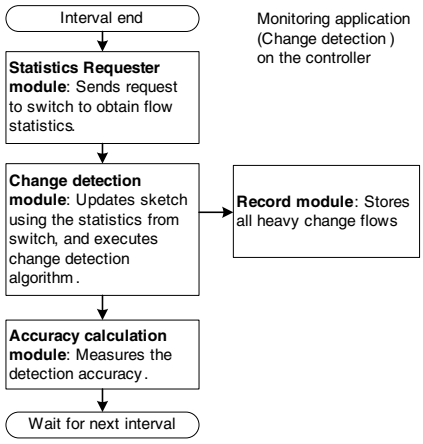

Fig. 2. Change detection system workflow.

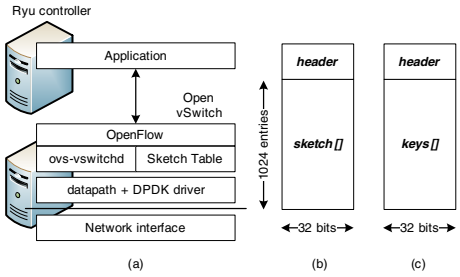

Fig. 3. (a) Sketch in switch+DPDK architecture. (b) OFPT_SKETCH_REPLY message format. (c) Key list.

(standard or parallel). The system then waits for the end of the observation interval. When the observation interval ends, it instructs the statistics requester module to send the flow statistics request message and obtain the flow statistics from all switches. After obtaining the flow statistics, the data are processed according to the administrator settings. The flow statistics are used to update the sketch inside the change detection module. The change detection module is then called to update the sketch and performs the change detection process. The source IP address and byte counts are used as the key and the updated value for the sketch. The change detection module implements the sketch-based change detection algorithm based on [21]. It creates the observed sketch $S_o(t)$ in each observation interval, calculates the forecast sketch $S_f(t)$, forecast error sketch $S_e(t)$, and then determine the alarm threshold T_A. The forecast error sketch $S_e(t)$ is queried to determine whether a source IP address is a heavy changer. If the estimated value for a source IP address returned by the query to $S_e(t)$ is larger than a threshold T_A, then the source IP address is added to a list by the record module. The controller can use this information to instruct the switch to drop the heavy changers. The accuracy calculation module then calculates the accuracy of the change detection and presents the results to the system administrator.

B. In-switch Sketch Implementation

The detection accuracy in the typical OpenFlow approach for traffic monitoring is limited by the TCAM size that collects the flow statistics. Even by utilizing several switches in parallel such that the change detection system can obtain more statistics, the scalability and flexibility are still the problems to be solved. Therefore, as shown in Fig. 3, we proposed an alternative implementation by using the sketch data structure as a monitoring primitive inside the Open vSwitch. This implementation also reduces the controller workload since the sketch update operation is done in the switch. The controller only needs to collect the sketch data structure from switches and perform the operations of change detection.

To collect the sketch from a switch to the controller, the OpenFlow 1.0 protocol is extended with two new messages of OFPT_SKETCH_REQUEST and OFPT_SKETCH_REPLY. The first message is sent by the controller to a switch requesting the sketch. The reply message is used by a switch to transport the sketch table back to the controller. Fig. 3(b) shows the OFPT_SKETCH_REPLY message structure. During operation, at the end of each observation interval, the controller sends the OFPT_SKETCH_REQUEST message to the switch and then wait for the OFPT_SKETCH_REPLY message. On the switch side, if the switch receives such request, it sends back the OFPT_SKETCH_REPLY messages contains the sketch table to controller. The communication flows between a controller and a switch is shown in Fig. 4.

A list of the flow key (i.e., the source IP address, destination IP address, 5-tuple, etc.) is also sent to the controller along with sketch reply message. The key is required to query the sketch for heavy changers.

In an observation interval, the number of keys can be enormous. As an example, in a 30 seconds interval, the number

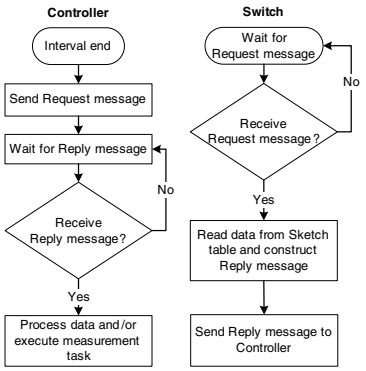

Fig. 4. Controller and switch message exchange flow.

of distinct source IP addresses in the trace that is used in our experiments is around 9,600. The total size of all these keys is 37.5 KBytes.

Typically, not all of the keys are needed. This is because we only interested in suspicious keys that are the heavy changers. In the proposed system, either limited number of keys or all keys can be sent to the controller. In the current implementation, the number of keys sent to the controller is limited to only 1,024. Sampling and filtering mechanism can be applied to select only the potential heavy changers so that a limited number of keys can be sent. For example, a simple filtering can be utilized to avoid duplicated keys after sampling. On the other hand, if all keys are required to be sent to the controller, multipart messages can be used. Carrying only a small number of keys can yield inaccurate detection results. Therefore, the strategy of keys selection is needed to avoid low accuracy detection results.

IV. Evaluation

The evaluation of the typical implementation is presented first followed by the in-switch sketch implementation.

A. Performance of the Typical Implementation

The test-bed, for the evaluation of the typical implementation, consists of a Floodlight controller and several hardware OpenFlow switches (Edge-Core AS4610-30T). The 15-minute MAWI network traffic traces [23], shown in Table I, are used for the evaluation. The trace is a part of 24h-long trace in the transit link of WIDE to the upstream ISP that is collected at MAWI's samplepoint-F. Figure 1 presents several configurations of change detection. The standard mode, parallel mode, and sampling mode are evaluated in the experiment.

The observation interval is 30 seconds. The depth (K) and width (H) of the sketch are set as $H = 5$ and $K = 2^{16}$ in all experiments. The false negative rate (FNR) and false positive rate (FPR) are used as accuracy metrics of the change detection. The FNR is defined as the unidentified heavy changers and the FPR is the non heavy changer that are incorrectly identified as heavy changers.

TABLE I
The information of network traffic traces [23] used in the evaluation.

Trace name	200701011400	200901091400
# of packets	6,948,502	20,518,350
# of distinct srcIP address	79,823	149,679
# of distinct srcIP (30s interval)	9,638 (avg.)	14,688 (avg.)

TABLE II
Detection error rate in different configurations [24].

Trace/Mode		Standard	Parallel	Sampling
Trace 1: 200701011400	FNR	24.18%	6.35%	32.79%
	FPR	0.54%	0.19%	0.80%
Trace 2: 200901091400	FNR	31.42%	19.64%	42.14%
	FPR	0.45%	0.28%	0.72%

B. Experimental Results and Discussion

The experimental results are shown in Table II. Trace 1 has lower FNR than Trace 2 in all modes. This is because Trace 2 has nearly twice the number of distinct source IP addresses than those in Trace 1. Because of the limited switch processing capability and resources, a large number of distinct source IP addresses cannot be processed. The switch loses some flows and makes the system unable to detect the heavy changers accurately. The FPR in all modes is roughly below 1%. Based on the observation in all experiments, the main factor that influences the FNR is the table capacity to handle the traffic. On the other hand, the FPR is affected by the sketch's depth H and width K. These parameters determine the amount of memory used by the sketch. The parallel mode is superior to the standard mode because in parallel mode, hardware resources of multiple switches are used. The system is able to process more flows. The sampling approach is also implemented for comparison purpose. In this approach, packets are sampled before being updated in the sketch. The sampling rate is chosen based on the recommended sampling rate [25]. The sampling rates for Trace 1 and Trace 2 are 1/300 and 1/550, respectively. The experimental results showed that the proposed system is superior compared to the sampling approach. As shown in Table II, the FNR of the sampling approach increased as the traffic size increased.

To understand the relationship between the capacity of the hardware switch and the change detection accuracy, experiments using standard mode and different *IdleTimeOut* values are conducted. Trace 1 is used in the experiments and the observation interval is set to 30 seconds. A commodity hardware OpenFlow switch has a limited CPU capacity and TCAM size. If there are too many flows need to be forward to the controller, the switch's CPU loading increase and traffic can not be processed properly. Limited TCAM size also affects the change detection accuracy. There are three factors that can contribute the CPU load as follows.

1) TCAM Capacity: The size of TCAM in the AS4610-30T switch can store up to 4,096 flow entries. The flow entries will be allocated in DRAM if more than 4,096 flow rules are used. Because the DRAM access time is much longer than that of

the TCAM, the switch CPU spends more time on flow lookup and matching process causing longer latency.

2) Flow Entry: Removal of a flow entry is determined by the *IdleTimeOut* parameter. If the number of flow entries to be removed is too large, the removal process takes longer time and increases the CPU load. In the experiments, we found that when the switch CPU is busy, it cannot remove the flow entries accurately according to the *IdleTimeOut* setting. These flow entries that are still stored in the flow table cause extra burden to system.

3) Generation of PACKET_IN messages: In the system, installation of flow entries to the switch's flow table is depend on the PACKET_IN message. If the number of flow entries is too large, the switch CPU will be busy generating PACKET_IN message, and the switch cannot handle the incoming traffic properly.

The experiment results are shown in Fig. 5. Four values are observed in their relation to the *IdleTimeOut* parameter: 1) Average flow per interval, 2) Distinct source IP loss rate, 3) Byte loss rate, and 4) False negative rate. By obtaining the average flow per interval, the system is able to to estimate how many flow entries should be stored in TCAM and DRAM, and reflects the switch CPU load. In Trace 1, there are 79,823 and about 9,638 distinct source IP addresses in 15 minutes and 30 seconds interval, respectively. The system uses the number of PACKET_IN messages sent from switch to controller and compares it with the number of distinct source IP addresses to estimate how many flow entries it needs to handle. As shown in Fig. 5, with the increasing of *IdleTimeOut*, the byte loss decreases. The reason is that the majority of flows have their corresponding flow entries in the flow table and the switch does not have to wait for processing a new flow entry. Therefore, it can forward the packet directly which makes the byte loss rate decreases. As the most important indicator of the system, the FNR has highest value when the *IdleTimeOut* is small. The main reason for this is that there is a part of flow entries has been removed by the switch before the end of the observation interval, making the detection inaccurate. In addition, the FNR is low when the *IdleTimeOut* is set to 60 seconds, and then it show an increasing trend. This is because the number of flow entries in flow table are too large for the switch CPU to maintain, hence decreasing the accuracy rate.

In summary, the switch CPU load is influenced mainly by the flow entries addition/deletion process and the generation of PACKET_IN messages. Maintaining the flow entries that are not stored in the TCAM is relatively has lower effect to switch CPU load. By increasing the *IdleTimeOut* parameter, the switch has to maintain more flow entries, but reduces the load to maintain the flow entries addition/deletion and PACKET_IN generation at the same time, resulting in higher detection accuracy.

C. Performance of the In-switch Sketch Implementation

This implementation is evaluated based on the same MAWI traces. The network setup in the experiment is shown in Fig. 6. Commodity PCs (Intel Core-i7-2600, 4GB DDR3 DRAM)

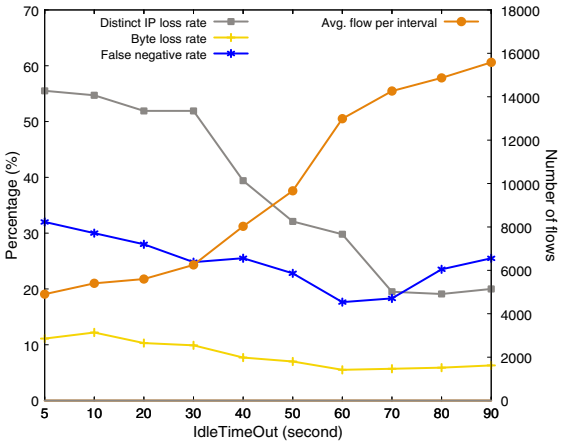

Fig. 5. The performance evaluation results with different *IdleTimeOut* values [24].

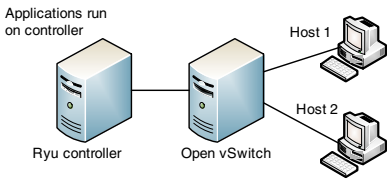

Fig. 6. Network setup used in the experiment for *in-switch* sketch implementation.

1GbE equipped with 10GbE network interface cards are used for the Open vSwitch and the controller. The sketch data structure is implemented in the user-space of the Open vSwitch. The sketch update process is performed in parallel with the forwarding path, therefore, the impact on the forwarding latency and throughput is minimal.

The performance evaluation is focused on the CPU loading and the traffic overheads between the controller and the Open vSwitch. The *simple switch,* provided by the Ryu controller, is used as the base-line for the comparisons as no sketch request, reply messages and flow statistics are sent between the controller and the switch. The traffic overhead between controller and switch is measured while replaying the MAWI network traffic trace files.

In the experiment of the Sketch_OF approach, the OFPT_SKETCH_REQUEST message is sent by the controller in a 30-second interval to the switch. The controller also processes the OFPT_SKETCH_REPLY message sent from the switch. The size of the sketch and key list is 12 KB ($3 \times 1,024$ counters) and 4 KB (1,024 keys), respectively.

In the typical approach, the controller sends request to get the flow statistics information from the switch at the end of each 30-second observation interval.

As shown in Table III, the traffic overheads in the typical implementation are much larger than that of the *simple-switch*. This is because in a typical implementation, a large number

TABLE III
THE INCREASES OF TRAFFIC OVERHEADS BETWEEN THE CONTROLLER AND OPEN VSWITCH. THE OPERATION OF THE *Simple-Switch* IS USED AS THE BASED-LINE FOR COMPARISONS.

Test mode/Trace name	200701011400	200901091400
Sketch_OF	17%	35%
Typical Implementation	more than 100%	more than 100%

TABLE IV
AVERAGE CPU LOADING IN THE OPERATION.

Test mode/Trace name	200701011400	200901091400
Simple Switch	1%	5%
Sketch_OF	19%	20%
Typical Implementation	29%	83%

of distinct IP addresses causing many PACKET_IN and PACKET_OUT messages in the link between the controller and the switch. The monitoring rules also occupy the space of the flow table designed originally for the forwarding purpose. Therefore, the total traffic throughput is affected with high packet loss rate. Based on the observation in the experiments, the packet loss rate in the typical implementation can be up to 73% and 85% for Trace 1 and Trace 2, respectively. On the other hand, in both *simple-switch* and Sketch-OF modes, no packet loss occurs. The operation of Sketch-OF mode is basically the same as that in the *simple-switch* mode except the extra sketch request and reply messages. Therefore, it does not have too much overhead compared to the typical implementation as shown in Table III.

The CPU load is measured by using the *top* command in Linux. As shown in Table IV, the highest average CPU load occurs in normal mode. The average CPU loads are 29% and 83% for Trace 1 and Trace 2, respectively.

V. CONCLUSION AND FUTURE WORK

This paper presents an implementation of a sketch-based traffic change detection system in SDN. The proposed system can work in commodity hardware OpenFlow switches. The operations can be performed in a *standard mode* that uses only one switch or in a *parallel mode* that uses more than one switches to increase the detection accuracy. The implementation of sketch data structure inside an Open vSwitch supporting traffic monitoring is developed. The OpenFlow protocol is extended with new messages to handle the sketch data processing between switches and the controller.

Evaluation using real network traffic traces are used to verify the proposed system. The experiment results show that the proposed system can detect the change in network traffic more accurate than the sampling approach.

For future work, we plan optimize the system to obtain higher detection accuracy. A mechanism to select only flows that have the potential to be the heavy changers will be developed. Filtering mechanism can be developed for eliminating non-heavy-changer flows to save the switch TCAM resource. Regarding the in-switch sketch implementation, the performance of the proposed system can be enhanced by using the DPDK library [26]. The packet processing flow can be optimized by using batch processing, huge-pages, bypassing the OS kernel, and avoiding unnecessary packet copying in the Linux operating system. Therefore, the time needed for packets to travel from the network interface to the CPU is significantly reduced, achieving the wire-speed packet processing at high speed.

ACKNOWLEDGMENT

This research was funded in part by the Ministry of Science and Technology, Taiwan, under contract number: MOST 106-2221-E-033-009 and MOST 105-2632-E-033-049.

REFERENCES

[1] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," White Paper, 2012.

[2] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 29–42.

[3] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches," in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, ser. Hot-ICE'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 13–18.

[4] M. Moshref, M. Yu, and R. Govindan, "Resource/accuracy tradeoffs in software-defined measurement," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 73–78.

[5] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "DREAM: Dynamic resource allocation for software-defined measurement," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 419–430.

[6] N. L. M. v. Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–8.

[7] D. Hamad, K. Yalda, and I. Okumus, "Getting traffic statistics from network devices in an sdn environment using openflow," in *Information Technology and Systems (ITaS)*, Sep. 2015, pp. 7–11.

[8] Y. Zhang, "An adaptive flow counting method for anomaly detection in SDN," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: ACM, 2013, pp. 25–30.

[9] J. SuÃ¡rez-Varela and P. Barlet-Ros, "Towards a NetFlow Implementation for OpenFlow Software-Defined Networks," in *2017 29th International Teletraffic Congress (ITC 29)*, vol. 1, Sep. 2017, pp. 187–195.

[10] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon," in *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 101–114. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934906

[11] Y. Yu, C. Qian, and X. Li, "Distributed and collaborative traffic monitoring in software defined networks," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 85–90. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620739

[12] T. Wellem, Y.-K. Lai, C.-H. Cheng, Y.-C. Liao, L.-T. Chen, and C.-Y. Huang, "Implementing a heavy hitter detection on the NetFPGA OpenFlow switch," in *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, Jun. 2017, pp. 1–2.

[13] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, April 2005.

[14] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 117–130. [Online]. Available: http://dl.acm.org/citation.cfm?id=2789770.2789779

[15] P. Chaignon, K. Lazri, J. Francois, and O. Festor, "Understanding disruptive monitoring capabilities of programmable networks," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, Jul. 2017, pp. 1–6.

[16] L. Hendriks, R. Schmidt, R. Sadre, J. Bezerra, and A. Pras, "Assessing the quality of flow measurements from openflow devices," in *IFIP IMA*, Sep. 2016, pp. 1–8.

[17] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. Lakshman, "Application-aware Data Plane Processing in SDN," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 13–18.

[18] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming Platform-independent Stateful Openflow Applications Inside the Switch," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, Apr. 2014. [Online]. Available: http://doi.acm.org/10.1145/2602204.2602211

[19] A. Wang, Y. Guo, F. Hao, T. V. Lakshman, and S. Chen, "UMON: Flexible and Fine Grained Traffic Monitoring in Open vSwitch," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '15. New York, NY, USA: ACM, 2015, pp. 15:1–15:7. [Online]. Available: http://doi.acm.org/10.1145/2716281.2836100

[20] X. T. Phan and K. Fukuda, "Sdn-mon: Fine-grained traffic monitoring framework in software-defined networks," *Journal of Information Processing*, vol. 25, pp. 182–190, 2017.

[21] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: Methods, evaluation, and applications," in *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement, IMC '03*. ACM, October 2003, pp. 234–247.

[22] Big Switch Networks, "Floodlight OpenFlow Controller," 2012. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[23] MAWI Working Group, "MAWI Working Group Traffic Archive," http://mawi.wide.ad.jp.

[24] Zong-cheng Liu, "An implementation and design of heavy change detection system based on commodity openflow switches," CYCU, 2016, MS Thesis.

[25] P. Phaal, "sFlow sampling rates," 2009. [Online]. Available: http://blog.sflow.com/2009/06/sampling-rates.html

[26] Intel Corp., "Data Plane Development Kit." [Online]. Available: http://www.dpdk.org/