Adaptive and Distributed Monitoring Mechanism in Software-Defined Networks

Xuan Thien Phan
The Graduate University for Advanced Studies
Tokyo, Japan
Email: thien@nii.ac.jp

Ignacio Dominguez Martinez-Casanueva
Technical University of Madrid
Madrid, Spain
Email: i.dominguezm@dit.upm.es

Kensuke Fukuda
National Institute of Informatics
Tokyo, Japan
Email: kensuke@nii.ac.jp

***Abstract*—Network traffic monitoring is an important factor to ensure the controllability and manageability of software-defined network (SDN). The current monitoring mechanism of SDN requires switches to request the controller for instructions to install flow entries for every new incoming flow. For fine-grained monitoring, which requires many flow entries in switches' flow tables, this mechanism creates a non-trivial delay in the forwarding of switches and overhead in the control channel. Our previous work presented SDN-Mon, a monitoring framework that supports fine-grained monitoring for SDN. In this paper, we discuss the aspect of monitoring the flows in a distributed manner. We believe that a distributed monitoring capability enhances the monitoring scalability for SDN. We propose a mechanism that supports SDN to distribute the monitoring load over multiple switches in the network, in which it prevents flows monitoring duplication and balances the monitoring load over switches in the network. With the proposed mechanism, each switch handles much less monitoring load; and the overhead at switches, the control channel, and the controller caused by the monitoring duplication is eliminated. We implement the proposal and integrate it to SDN-Mon to enable a scalable and distributed monitoring capability in SDN. Experimental results show that the proposed mechanism significantly reduces the amount of monitoring load per switch, while the monitoring load is well balanced over switches in the network, with only an acceptable polling and processing overhead.**

***Index terms*—** Software-Defined Networking; OpenFlow; Network monitoring; Load balancing;

I. Introduction

Network traffic monitoring is a key factor for effective network control and management in Software-Defined Network (SDN) [1] [2] [3]. It provides network information that is critically important for the network intelligence to ensure the stability, availability, and security as well as maintain other SDN-provided benefits for network services and applications. However, the current monitoring mechanism of SDN requires switches to report to the controller, e.g. by sending packet-in messsages, and wait for the controller's instructions for every new flow. For monitoring a large number of flows, this approach cannot adapt well because it creates non-trivial delay in the forwarding process of the switches, and overhead in the control channel and the controller due to the frequent reports and responses between switches and the controller. Different from that approach and existing works that rely on the same monitoring mechanism, our previous work, i.e. SDN-Mon [4], presented an approach where switches actively monitors flow without interrupting the controller. This approach introduces no delay in forwarding process of switches or overhead in the control channel and the controller for scalable monitoring capability for SDN.

This paper discusses extending the SDN-Mon approach to support monitoring on multiple switches for a network-wide monitoring solution for SDN. We propose a mechanism that supports SDN to distribute the monitoring load of the whole network to multiple switches in the network, where monitoring duplication is detected and eliminated, and the amount of monitoring loads to be assigned to switches are balanced. Different from existing works, which requires switches to interract with the controller for every new flow, our mechanism introduces a "lazy approach" where switches can actively monitor flows without interrupting the controller. Then, the task of balancing the load in switches and rejecting the duplicated flows will be processed only in every monitoring data querying time. With the proposed mechanism, the delay in the switches' forwarding process and the overhead on the controller and the control channel are decreased since the frequency of query is much less than the frequency of new flow.

We designed and implemented the proposed mechanism on SDN-Mon to enable a fine-grained and scalable network-wide monitoring capability in SDN. We evaluated the efficiency of the proposal in terms of reducing the number of monitoring entries in switches, and balancing the monitoring load/monitoring entries assigned to the switches. The experimental results show that the proposed mechanism largely reduces the number of monitoring entries stored in each switch (up to 63% for a network with three switches), and the monitoring entries are assigned to the switches in a balancing way, with only an acceptable overhead for querying and processing even hundreds of thousands of monitoring entries.

II. Related Work

Our work is motivated by various existing approaches for network monitoring in SDN. A number of these approaches, i.e., Avant-guard [5], OFX [6], DevoFlow [7], UMON [8], OpenSketch [9], and Payless [10], propose extensions to the SDN data plane for improving performance of SDN monitoring. These approaches mainly discuss monitoring in a single switch. Some other proposals use network monitoring tools (e.g. sFlow [11]) to handle monitoring in SDNs instead of

978-3-901882-98-2 © 2017 IFIP

leveraging SDN switches [12] [13]. The disadvantages of these approaches are that they are not integrated with OpenFlow platform, and strictly require additional hardware deployment for flow collecting and analyzing systems.

Some other proposals discuss flow rules distribution on multiple switches. OpenTM [14] proposes a switch selection mechanism for querying flow statistics to estimate traffic matrix. OpenWatch [15] distributes flow entries over switches to reduce the number of flow entries per switch. Similarly, FlowCover [16] proposes a mechanism for polling-switches selection and flow-statistics aggregation to reduce monitoring overhead. LiteFlow [17] proposes a selection scheme at a controller that chooses a switch to monitor all flows between an end-hosts pair, while leaving other path-switches forwarding packets without any monitoring process. These approaches basically monitor network traffic by installing flow entries in switches, which is not adaptable with a large number of flows. Moreover, those approaches require switches to request the controller and wait for its instructions for every new flow, which produces significant delay time in the forwarding processes of the switches, especially for busy networks with large frequency of new flows. Another approach [18] uses two-stage Bloom filters for flow-size counting and packet sampling, and distributes monitoring load over all switches in the network. This approach totally relies on Bloom filters for monitoring that shows certain false positive rate, and deleting an entry in that data structure inflexibly requires reconstruction of the Bloom filters.

III. Architectural Design of Extended SDN-Mon

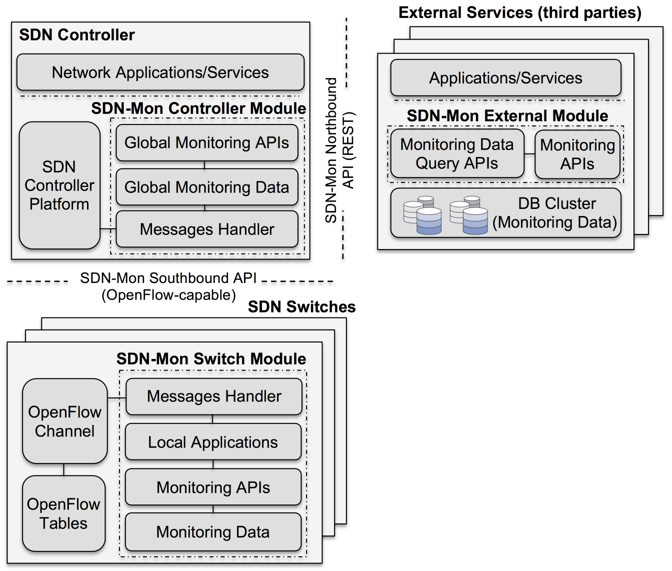

Fig. 1: Extended SDN-Mon architecture.

Our previous work introduced SDN-Mon [4], a scalable monitoring framework that supports fine-grained traffic monitoring for various network applications and services. In this paper, we propose a mechanism that supports distributed monitoring capability (i.e. by using multiple SDN switches) for SDN, and intergrate it to SDN-Mon (called extended SDN-Mon). Fig. 1 illustrates the architecture of extended SDN-Mon which support monitoring with multiple switches and external applications and services of third parties.

The extended SDN-Mon consists of three major modules: the switch module, the controller module, and the external module. The switch module handles the monitoring functionality at a switch [4], while the controller module provides global monitoring APIs and global monitoring data for controller applications and services. The external module provides monitoring data query APIs that support third parties to remotely extract the global monitoring data from the controller, and monitoring APIs that correspond to the monitoring APIs at the controller module for external applications and services. Each flow that traverses through a switch is monitored by a *monitoring entry* (or *m-entry*), which consists of monitoring match fields (e.g. 5-tuple), counters, and the updating timestamp, in the switch's monitoring database. The proposed mechanism in this paper enables SDN-Mon to automatically assign the monitoring load to multiple monitoring switches in a balanced way. This allows network operators to leverage SDN-Mon APIs and the global monitoring data for their applications without any further effort for managing the multiple monitoring switches.

IV. Distributed Monitoring Mechanism for SDN

A. Organization of Global Monitoring Data

In the proposed mechanism, we organize the global monitoring data at the controller into *Global Monitoring Tables (GMTs)*; each corresponds to each monitoring switch in the network. Each table stores, updates, and manages the monitoring data (i.e., m-entries) of a switch. Each m-entry includes following fields: *Monitoring match fields*, *Counters*, *Last update*, and *Hash*. The hash value of each entry is unique, and a corresponding hash-based data structure is used for fast lookups and other data based processes of *GMTs*. The m-entries in each *GMT* is kept updated by the controller through frequently polling of m-entries at the corresponding switch.

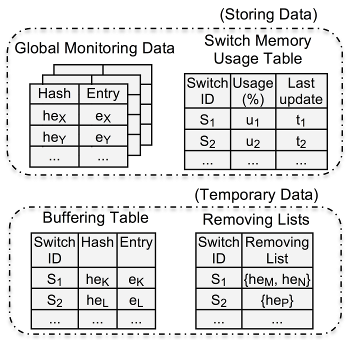

Fig. 2: Organization of the global data.

A lightweight table called *Switch Memory Usage Table* (Fig. 2) holds frequently updated information of the memory usages of switches for monitoring load balancing purpose. Each entry in this table holds the memory usage information of a certain switch, including the following fields: <*switch ID, Usage,* *Last update*>. The *switch ID* is the identification number of the switch. The *Usage* represents the percentage of used memory: *Usage* $= (N_{fe} + N_{me})/Capacity$, where $Capacity$ is approximately estimated by the maximum number of flows and m-entries that the switch can handle (based on the switch configuration), N_{fe} and N_{me} are the current numbers of flow entries and m-entries at the switch. The *Last update* is the timestamp of the latest update of N_{fe}, N_{me} and *Usage*.

Besides the *GMTs*, the controller also contains temporary data structures consisting of a *Buffering Table* and *Removing Lists*. The buffering table holds selected m-entries from lists of m-entries that the controller receives for each monitoring data query. Removing lists (*RLs*), in which each one corresponds to each monitoring switch, hold hashes of rejected m-entries. These m-entries are duplicated ones that are not chosen from the switch selection process. When the controller completes processing the received monitoring data of each querying time, the m-entries in the removing lists will be physically removed from that switch, and the buffering table and the removing lists will be cleared for processing the received data of the next querying time.

B. Workflow of the proposed mechanism

For every query-time-interval, the controller sends *SDN-Mon Data Request* messages to all monitoring switches to query the new and updated m-entries at switches. Each switch responds such request by sending a *SDN-Mon Data Reply* message including a list of new m-entries and updated ones (the m-entries whose counters have been updated since the previous query of the controller) to the controller (Fig. 3). A *SDN-Mon Data Reply* also includes N_{fe} and N_{me}, which are counted by the SDN-Mon switch module, for updating the memory usage information of that switch in the *Switch Memory Usage Table* at the controller. For a flow that traverses through multiple switches in network, such switches may actively install m-entries to sample/monitor that flow (the probability of sampling the flow is based on the sampling ratio), resulting in the duplicated m-entries stored at the switches.

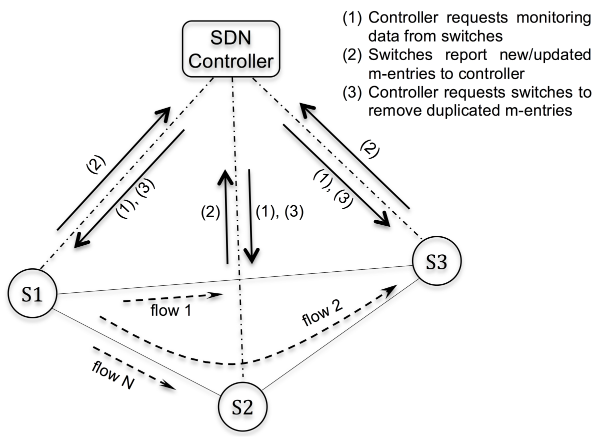
(1) Controller requests monitoring data from switches
(2) Switches report new/updated m-entries to controller
(3) Controller requests switches to remove duplicated m-entries
SDN Controller
(2)
(1), (3)
(2)
(1), (3)
(2)
(1), (3)
S1
S2
S3
flow 1
flow 2
flow N

Fig. 3: Controller-switches communication.

Upon receiving the m-entries from switches, the controller filters the m-entries by: selecting a switch with smallest memory usage to keep monitoring a m-entry for each flow, putting the hashes of the duplicated m-entries to *RLs* for later removal at switches. For switches whose removing list is not empty, the controller sends instructions to them to physically remove the rejected entries after processing all received m-entries of a querying time. Algorithm 1 illustrates the detailed steps for processing the received monitoring data of a querying time at the controller.

```
Data: Lists of m-entries received from switches
for Each list of m-entries $L_i$ from a switch $S_I$ do
    for Each m-entry $e_X$ in $L_i$ do
        if $e_X$ is an updated m-entry then
            Update the corresponding m-entry in Global Monitoring
            Table of $S_I$;
        else
            if $e_X$ is not existed in Buffering Table then
                Insert $e_X$ into Buffering Table;
            else
                Assume $e_Y$ is the duplicated m-entry, $S_{e_Y}$ is
                corresponding switch of $e_Y$;
                if switch-usage($S_I$) < switch-usage($S_{e_Y}$) then
                    Insert $e_X$ into Buffering Table;
                    Insert $e_Y$ into Removing List of $S_{e_Y}$;
                    Update Switch Memory Usage Table;
                else
                    Insert $e_X$ into Removing List of $S_I$;
                end
            end
        end
    end
end
```

Algorithm 1: Pseudocode for processing the received m-entries of a querying time at the controller.

V. Evaluation

We evaluate the effectiveness of the proposed mechanism in three aspects: the reduction of the per-switch monitoring rules storage, the balance in the number of monitoring rules that are distributedly assigned to the switches, and the overheads (elapsed times) of the proposed mechanism and the monitoring system.

A. Experiment environment

We conducted experiments with a virtual SDN deployment using VNX [19], as illustrated in Fig. 4. The controller is run on PC-1, a physical computer with a 2.66 GHz CPU Intel core 2 Duo E6750 (2 cores) and 4 GiB RAM. Three switches and three hosts are run on PC-2, another physical computer with a 3.4 GHz CPU Intel core i7 (8 cores) and 8 GiB RAM. PC-1 and PC-2 are connected via a LAN network. The switches are run with DPDK v16.11 [20] to enhance its processing speed.

B. Evaluation on the reduction of monitoring load per switch and the monitoring load balance among multiple switches

The monitoring match fields in SDN-Mon switch module are set with 5-tuple consisting of source IP address, source port, destination IP address, destination port, and protocol. The query time interval is set to 10 seconds, which means the controller queries new and updated m-entries from switches in every 10 seconds. Sampling ratio is set to 1.0. We conduct

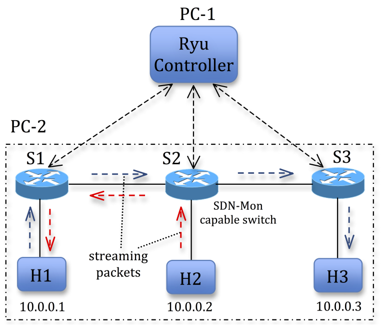

Fig. 4: Experiment network.

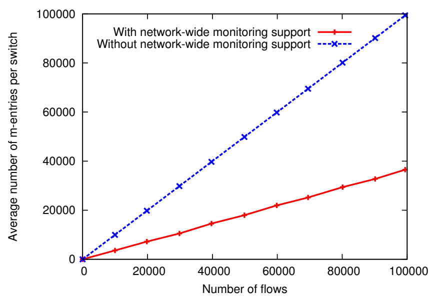

Fig. 5: Average number of m-entries per switch in various number of flows.

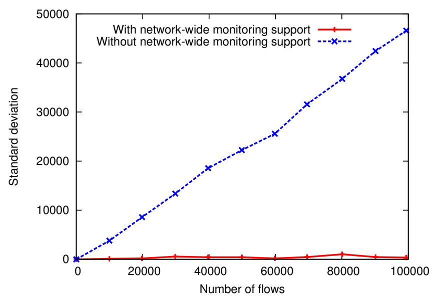

Fig. 6: Standard deviation of numbers of m-entries in switches.

the experiments for two cases: monitoring with the support of our proposed mechanism, and monitoring without the support of our proposed mechanism.

For evaluating the reduction of monitoring load per switch, we inject a stream of packets from *H1* to *H3* using *tcpreplay* [21]. The dataset for packet injecting is a pcap file from MAWI traffic repository [22], which is a 6.6 GB file of network traffic captured from a real backbone network in Feb. 26, 2017. We measure the number of m-entries installed in each switch in various numbers of 5-tuple flows (from 0 to 100,000 flows). The evaluation results (Fig. 5) shows that the average number of m-entries per switch is reduced as over 63%.

For evaluating the monitoring load balance among the switches, we inject two streams of packets from *H1* to *H3*, and from *H2* to *H1* concurrently with two pcap traces (of the same traffic volume), with the same packet injecting speed. We measure the numbers of m-entries in switches and calculate the standard deviation of these numbers. A small standard deviation means that the monitoring load balancing functionality works efficiently. The evaluation results (Fig. 6) shows that with the support of the proposed mechanism, the standard deviations are small for all different numbers of flows in the traffic. Thus, the switches are assigned with nearly equal or equivalent numbers of m-entries.

C. Elapsed times of the algorithm and the system

We evaluate the overhead of the proposed mechanism in two aspects: the elapsed time for processing the proposed algorithm (called the *algorithm elapsed time*), and the elapsed time for the whole process of a querying time (called the *system elapsed time*). The algorithm elapsed time is the amount of time for processing all arriving reply messages and the amount of time for putting the m-entries in the buffering table into the *GMTs* and removing the corresponding m-entries in the *RALs*. The system elapsed time is the time interval since the controller sends the first *SDN-Mon Data Request* message to a switch until it completes processing all *SDN-Mon Data Reply* messages received from switches.

We inject a pcap trace containing 634,500 5-tuple flows, which creates 232,950, 234,860, and 234,390 m-entries stored in the switches *S1*, *S2*, and *S3* respectively. We inject the packets at various injecting rates so that the numbers of m-entries that the controller receives in each query are from 20,000 to 100,000 m-entries. The experimental result (Table I) shows that both of the algorithm and system elapsed times are negligible, with small maximum values as 1.181 and 2.692 seconds correspondingly, even for processing a large number of m-entries as 100,000 m-entries per query, for a large number of 5-tuple flows as 634,500 flows in the injecting traffic.

Number of monitoring entries per query	20,000	40,000	60,000	80,000	100,000
Algorithm elapsed time (s)	0.237	0.436	0.742	0.939	1.181
System elapsed time (s)	0.371	0.868	1.361	1.966	2.692

TABLE I: Algorithm elapsed time and system elapsed time at various numbers of monitoring entries per query.

VI. Conclusion

In this paper, we propose an adaptive and distributed mechanism to support monitoring in multiple switches for SDN. The proposed mechanism distributes the monitoring entries over switches, selects switches to assign the monitoring tasks in a balanced fashion, and eliminates duplicated monitoring entries. Thus it non-trivially reduces the amount of monitoring overhead in switches, in the controller and the control channel. We integrated the proposed mechanism on SDN-Mon, a fine-grained monitoring framework, to enable a scalable and distributed monitoring capability in SDN. Our performance evaluation with real traffic data shows a significant reduction of monitoring load in each switch, and an efficient balance of monitoring load among switches. The elapsed times of the proposed mechanism and the monitoring system are also proved to be small, as less than one and three seconds respectively, for handling a large number of entries per query as a hundred thousand, with a large number of flows as over a half million in the network traffic.

References

[1] Open Networking Foundation, “Software-defined networking: The new norm for networks,” *ONF White Paper*, vol. 2, pp. 2–6, 2012.

[2] S. Sezer, S. Scott-Hayward, P.-K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, “Are we ready for SDN? Implementation challenges for software-defined networks,” *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[4] X. T. Phan and K. Fukuda, "SDN-Mon: Fine-grained traffic monitoring framework in software-defined networks," *Journal of Information Processings (JIP)*, vol. 25, pp. 182–190, Feb. 2017.

[5] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *ACM CCS'13*, 2013, pp. 413–424.

[6] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith, "Enabling practical software-defined networking security applications with OFX," *NDSS'16*, pp. 1–15, 2016.

[7] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," in *ACM SIGCOMM CCR*, vol. 41, no. 4, 2011, pp. 254–265.

[8] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "UMON: Flexible and fine grained traffic monitoring in Open vSwitch," *ACM CoNEXT'15*, pp. 1–7, 2015.

[9] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch." in *NSDI'13*, 2013, pp. 29–42.

[10] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *IEEE NOMS'14*, 2014, pp. 1–9.

[11] P. Phaal, S. Panchen, and N. McKee, "InMon Corporation's sFlow: A Method for Motoring Traffic in Switched and Routed Networks," RFC3176, 2001.

[12] A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou, "OrchSec: An orchestrator-based architecture for enhancing network-security using network monitoring and SDN control functions," in *IEEE NOMS'14*, 2014, pp. 1–9.

[13] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "OpenSample: A low-latency, sampling-based measurement platform for commodity SDN," in *IEEE ICDCS'14*, 2014, pp. 228–237.

[14] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for OpenFlow networks," in *PAM'10*, 2010, pp. 201–210.

[15] Y. Zhang, "An adaptive flow counting method for anomaly detection in SDN," in *ACM CoNEXT'13*, 2013, pp. 25–30.

[16] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "FlowCover: Low-cost flow monitoring scheme in software defined networks," in *IEEE GLOBECOM'14*, 2014, pp. 1956–1961.

[17] N. Grover, N. Agarwal, and K. Kataoka, "liteFlow: Lightweight and distributed flow monitoring platform for SDN," in *IEEE NetSoft'15*, 2015, pp. 1–9.

[18] Y. Yu, C. Qian, and X. Li, "Distributed and collaborative traffic monitoring in software defined networks," in *ACM HotSDN'14*, 2014, pp. 85–90.

[19] Virtual Networks over Linux (VNX), "http://web.dit.upm.es/vnxwiki/index.php."

[20] DPDK: Data plane development kit, "http://dpdk.org/."

[21] Tcpreplay, "http://tcpreplay.synfin.net/wiki/tcpreplay."

[22] MAWI Traffic Repository, "http://mawi.wide.ad.jp."