Electronic Preprint for Journal of Information Processing Vol.25

Regular Paper

SDN-Mon: Fine-Grained Traffic Monitoring Framework in Software-Defined Networks

Xuan Thien Phan[1,a)] Kensuke Fukuda[1,2,b)]

Received: May 24, 2016, Accepted: November 1, 2016

Abstract: Fine-grained network traffic monitoring is important for efficient network management in software-defined networking (SDN). The current SDN architecture, i.e., OpenFlow, relies on counters in the flow entries of forwarding tables for such monitoring tasks. This is not efficient nor flexible since the packet-header fields that users aim for monitoring are not always the same or overlap with those in OpenFlow match fields, which is designed for forwarding as a higher priority. This inflexibility may result in unnecessary flow entries added to switches for monitoring and controller-switch monitoring-based communication overhead, which may cause the communication channel to become a bottleneck, especially when the network includes a large number of switches. We propose SDN-Mon, a SDN-based monitoring framework that decouples monitoring from existing forwarding tables, and allows more fine-grained and flexible monitoring to serve a variety of network-management applications. SDN-Mon allows the controller to define the arbitrary sets of monitoring match fields based on the requirements of controller applications to flexibly monitor traffic. In SDN-Mon, some monitoring processes are selectively delegated to SDN switches to leverage the computing processor of the switch and avoid an unnecessary overhead in the controller-switch communication for monitoring. We implemented SDN-Mon and evaluated its performance on Lagopus switch, a high-performance software switch.

Keywords: software-defined networks, OpenFlow, network monitoring

1. Introduction

Software-defined network (SDN) [11], [12], [18] is an emerging network architecture that decouples the control plane and the data plane for efficient and flexible network management. This network architecture provides a centralized control that allows network switches and routers to process network traffic in line-rate speed while its controller can manage the network in real time through a control channel. The most popular enabler of SDN is currently the OpenFlow platform [10], [12], which includes a well-defined protocol and OpenFlow-supported hardware switches/routers. SDN is aimed at enhancing the network control and management, which is beneficial for a variety of network applications. SDN has been being deployed more and more widely nowadays by research communities and industrial companies (e.g., Google deployed B4 [8], a software-defined WAN connecting Google's data centers across the planet). However, besides its benefits, SDN still has some drawbacks including: (1) overhead in monitoring at the controller due to the high frequency of queries for flow statistics; (2) inflexibility when monitoring is bound with forwarding into the same flow tables, while packet features for monitoring purposes are not always the same or overlap with forwarding purposes; and (3) lack of scalability due to the very limitted number of flow entries as current capacity support of hardware switches.

Using the current SDN platform for monitoring may cause a large amount of overhead due to the very frequent queries between the controller and switches. In particular, the current SDN-supported monitoring mechanism does not support sampling. With the current SDN/OpenFlow architecture, every first packet of a new flow arriving at a switch will cause the switch to send a packet-in message to the controller to request forwarding rules to process the packet. Even if the packet is not the first one of the flow (it may be one of the following packets of the same flow that are ignored by the controller based on requirements of monitoring-based applications or because of an overload problem), the switch also sends a new packet-in message to the controller to request instructions to process the packet. Such very frequent queries may cause a significant overhead to the controller-switch communication channel and consume more processing in the switches [5].

Moreover, monitoring based on the current SDN platform has little scalability. This problem is due to the fact that the maximum number of flow entries that current hardware switches can support is very limited (only a few thousands or a few tens of thousands depending on the configuration of matching fields), while the maximum number of flows in real networks can reach into the millions, much larger than the number of flows that a switch can handle. With those drawbacks, SDN monitoring is not as efficient as it should be to support various network monitoring functions for a variety of network management applications. We believe that without a more scalable and flexible framework for both forwarding and monitoring, it is not feasible for SDN to reach a wide deployment.

[1] SOKENDAI, Chiyoda, Tokyo 101–8430, Japan
[2] NII, Chiyoda, Tokyo 101–8430, Japan
[a)] thien@nii.ac.jp
[b)] kensuke@nii.ac.jp

Some SDN-based monitoring mechanisms and frameworks have been proposed recently for targeting specific applications but cannot provide a flexible platform to benefit a variety of monitoring applications [14], [25]. Another framework [26] is proposed that does not take into account the scalability issue of SDN-based monitoring, which results in a lack of scalability that limits the framework to be widely deployed. Other mechanisms are aimed at reducing the amount of communication overhead between the control plane and data planes [4], [21], [29], but still lack scalability and flexbility since their monitoring relies on unmodified flow tables without any sampling support.

To overcome those drawbacks, we propose a SDN-based monitoring framework called SDN-Mon for more fine-grained, flexible and scalable monitoring in SDN networks. With the support from SDN-Mon, the monitoring functionality is freed from the forwarding functionality. This allows the controller to operate the forwarding functionality in a more efficient and scalable way. For instance, by letting SDN-Mon handle monitoring, the controller can insert more general forwarding rules with wildcards into switches to properly forward flows instead of inserting very specific rules to handle both forwarding and monitoring. This helps decrease the total number of forwarding entries that a switch requires to process as well as saving processing power due to handling unnecessary forwarding rules. As a result, the forwarding functionality in a switch can be processed more efficiently while leaving more processing capability for more fine-grained, flexible, and scalable monitoring. With the above drawbacks in current SDN monitoring support, a scalable, flexible, and fine-grained monitoring framework like SDN-Mon is necessary for monitoring in SDN networks.

This paper makes the following contributions: (1) We propose SDN-Mon, a framework that separates the monitoring functionality from the forwarding functionality in an SDN switch, and allows monitoring to be processed in a more fine-grained, efficient way and independently from forwarding to serve a variety of SDN management applications; (2) We provide an architectural design for SDN-Mon and implement it on a Lagopus switch and a Ryu controller; (3) Our performance evaluation indicates that our monitoring framework incurs only an acceptable overhead in an SDN switch.

2. SDN-Mon Architecture

Our SDN monitoring framework (SDN-Mon) is a scalable framework that supports a fine-grained and flexible monitoring for controller applications. In SDN-Mon, the separation of monitoring and forwarding allows monitoring to be processed in a more fine-grained and independent manner from forwarding. SDN-Mon allows controller applications to monitor flows based on flexible monitoring match fields that are defined according to their monitoring demands. Unlike current frameworks, SDN-Mon supports an efficient sampling mechanism that makes it more scalable to be adaptable with larger-scale networks than current mechanisms. With SDN-Mon support, the SDN controller can determine forwarding logics in a more proper and scalable way with less flow entries in the switches' flow tables to avoid a possible overflow problem. Moreover, the number of controller-switch messages required for monitoring are smaller, this helps reduce the monitoring-based overhead in the SDN communication channel. SDN-Mon leverages the switches' processing power to process its monitoring functionality and is designed with a high priority for low processing overhead in a switch. In this section, we describe the architecture of SDN-Mon, how it operates alongside other processing of SDN switches as well as its APIs to support monitoring applications on SDN networks.

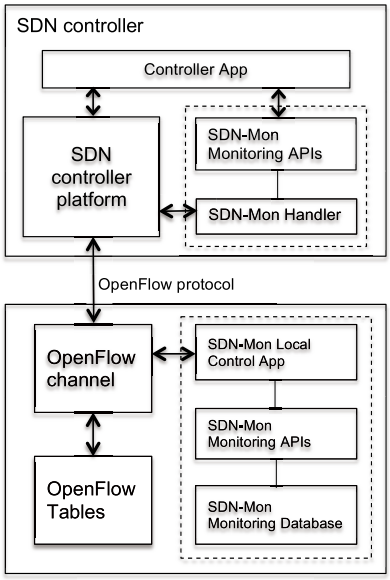

Fig. 1 SDN-Mon Architecture.

2.1 Architecture Overview

As shown in **Fig. 1**, SDN-Mon is composed of a controller-side module and a switch-side module. The controller-side module has SDN-Mon monitoring APIs that enable a flexible monitoring in the SDN controller. These APIs operate on top of the SDN controller platform to support monitoring purposes of controller applications. The switch-side module consists of three components: the SDN-Mon local control app, SDN-Mon monitoring APIs, SDN-Mon monitoring database. These components operate together to handle the monitoring functionality in switches. The support of these components together with the SDN-Mon monitoring APIs on the controller-side enables a more fine grained and flexible monitoring with a set of arbitrary monitoring match fields defined by controller applications.

2.1.1 SDN-Mon Monitoring APIs at Controller Side

The controller-side SDN-Mon monitoring APIs are programming APIs to support controller applications for monitoring. With these APIs, controller applications can request the switches to insert monitoring entries with a certain user-defined set of monitoring match fields, remove monitoring entries from the monitoring database in switches, query network statistics of monitoring entries in switches, as well as other monitoring control functions, as described in Section 2.3. These APIs serve a variety of monitoring purposes of controller applications independently from the forwarding functionality.

2.1.2 SDN-Mon Handler

The SDN-Mon handler is a module at the controller-side that

is responsible for handling SDN-Mon-related communication with switches. This module supports encapsulating the controller's monitoring request parameters into SDN-Mon messages and sending the messages to the switches. For SDN-Mon-related replies from the switches, the SDN-Mon handler supports parsing the received messages and extracting the monitoring data or switch notification information from the messages for controller applications.

2.1.3 SDN-Mon Local Control Application

The SDN-Mon local control app is a lightweight processing application that is responsible for handling SDN-Mon-related requests from the controller as well as local switch-management processing to ensure the proper operation of the SDN-Mon switch-side module. This application analyzes the monitoring-related messages delegated by the SDN communication channel between the control and data planes, which is enabled by the well defined OpenFlow protocol [12], and leverages the SDN-Mon monitoring APIs to process and manage the monitoring database based on the controller requests. For network-statistics requests from the controller, the SDN-Mon local control app queries the statistics from the monitoring database, encapsulates the statistical information, and delegates it to the controller-switch communication channel to send to the controller. The SDN-Mon local control app also reserves room for local pre-processing in switches for experimenters' applications (e.g., pre-checking flows in a switch for anomaly detection, and pre-checking flow volumes to detect large flows in the networks).

2.1.4 SDN-Mon Monitoring APIs at Switch

The switch-side SDN-Mon monitoring APIs are programming APIs for the SDN-Mon local control app to process the monitoring tasks delegated by the controller through the monitoring requests. Basically, the programming functions provided by these APIs correspond to the functions of the controller-side SDN-Mon monitoring APIs. These APIs support the SDN-Mon local control app to manage the SDN-Mon monitoring database and query monitoring statistics from the database to respond to the controller requests. Details of these APIs are described in Section 2.3.

2.1.5 SDN-Mon Monitoring Database

The SDN-Mon Monitoring Database includes a monitoring table and a Bloom filter. The monitoring table has a set of monitoring entries. Each monitoring entry consists of monitoring match fields, counters, and a hash value.

Since OpenFlow protocol [12] is basically the only protocol that enables SDN communication channel between the control and data planes, in our current design, monitoring match fields are a set of fields that correspond to the match fields of a flow entry defined in the OpenFlow protocol. These fields consist of a subset of or all OpenFlow match fields. Controller applications define monitoring match fields based on their monitoring purposes at the initial stage of the monitoring process. Counters consist of a list of packet counts and one of byte counts. These lists have an equal size (which is the number of packet count in the lists, called counters buffer size S_b) that is defined by controller applications at the initial stage of the monitoring process based on the applications' requirements. The packet and byte count lists

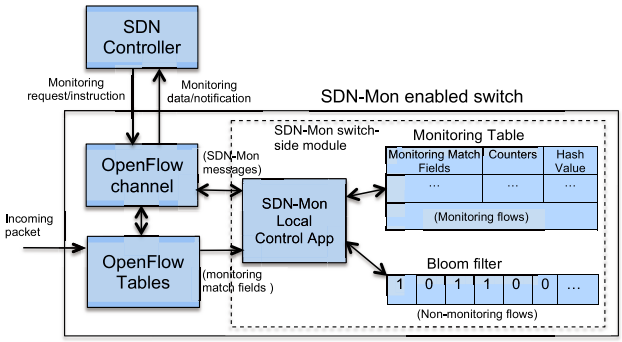

Fig. 2 Packet processing for monitoring at SDN-Mon enabled switch.

are buffers that record the historical packet and byte counts of a monitoring entry at every time interval ΔT_u. The counter-buffer-update time interval ΔT_u is calculated by the local control app based on S_b and a query time interval ΔT_q, which is also set by the controller at the initial stage of monitoring, with the fomula: $\Delta T_u = \Delta T_q / S_b$. These counter buffers support fine-grained monitoring and help decrease the number of necessary data exchanges between switches and controller for monitoring statistics. The hash value in a monitoring entry is calculated from the monitoring match fields of the entry. This hash value together with the hash table data structure implementation of the monitoring table support accelerating the lookup process in the monitoring table.

Due to the need of sampling in a variety of existing network monitoring applications, SDN-Mon provides a sampling mechanism in its monitoring process. This sampling mechanism is processed with the support of a Bloom filter [1], a lightweight data structure for checking the existence of certain entries. In SDN-Mon, the Bloom filter is used to mark specific flows corresponding to an incoming packet as non-monitoring to ignore them for monitoring based on the result of the sampling mechanism. This sampling mechanism allows the controller to control the total numbers of monitoring entries in a switch based on monitoring-based application requirements or to avoid possible overflow/overloading problems, by setting the sampling ratio in the SDN-Mon monitoring database via the SDN-Mon monitoring APIs. **Figure 2** illustrates in detail the components in the SDN-Mon switch-side module and how packets are processed in the module for monitoring.

2.2 Monitoring Process

For monitoring traffic at a switch, the controller first turns on the monitoring mode by sending a message to the switch to set the monitoring match fields. The monitoring process in SDN-Mon is initiated accordingly. When a packet arrives at a switch (where the monitoring mode is on), the pipeline packet processing of OpenFlow is performed. Concurrently, monitoring match fields are extracted from the packet-header fields and passed into the SDN-Mon local control app for monitoring. Based on the monitoring match fields, this application conducts a lookup process in the Bloom filter to verify if a corresponding entry exists in the Bloom filter. If the verification result is positive (this means that the entry based on these monitoring match fields is a non-monitoring entry), the SDN-Mon local control app ignores this

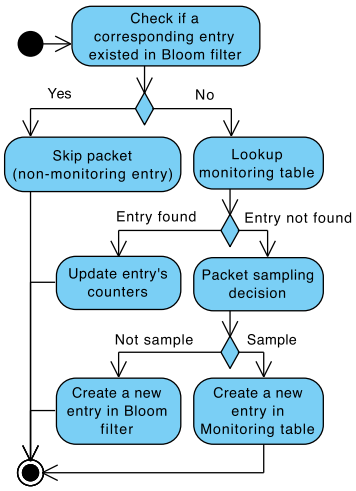

Fig. 3 Workflow of monitoring process in SDN-Mon switch-side module.

entry, and all following packets matching these monitoring match fields will not be monitored. If the verifying result is negative, the application conducts the lookup process in the monitoring table to find a monitoring entry whose monitoring match fields match the corresponding match fields in the packet.

If a match occurs, the counters in the matching monitoring-entry will be updated by increasing the packet count by one, and increasing the byte count by the number of bytes of the packet. If no matching entry is found, the SDN-Mon local control app determines whether to monitor this flow based on the sampling ratio set by the controller. If the flow is determined for monitoring, a new monitoring entry, in which the values of monitoring match fields are set by those of corresponding match fields of the flow, will be created and inserted into the monitoring table. If the flow is not determined for monitoring, a new entry/element, in which the values of monitoring match fields are set by the same way, will be created and inserted into the Bloom filter to mark the flow as non-monitoring. The workflow of the monitoring process is illustrated in **Fig. 3**. In SDN-Mon, the monitoring match fields are flexible and set by the controller based on the requirements of the controller applications. The controller can also insert specific monitoring entries for monitoring by using the SDN-Mon monitoring APIs to send monitoring entry modification requests to a switch to insert the entries.

2.3 SDN-Mon Monitoring APIs

The SDN-Mon monitoring APIs provide functions that the SDN-Mon controller-side and switch-side modules can use to exchange messages containing monitoring control instructions or monitoring data between each other in the OpenFlow communication channel. These functions allow the controller to define the set of monitoring match fields for the monitoring table in the switches, manage the monitoring table by setting a sampling ratio based on its requirements or monitoring table status, and query statistics of monitoring entries in switch-side monitoring tables. For the switches and the controller, SDN-Mon provides functions for adding a new monitoring entry into the monitoring table of a switch and removing an entry from the monitoring table. The functions of the SDN-Mon monitoring APIs are as follows.

sendToController (SDN-Mon message): This allows the switch-side module to send SDN-Mon messages to its controller. These messages include the Monitoring Statistics Reply Message and Overflow Notification Message.

sendToSwitch (SDN-Mon message, Switch ID): This allows the controller-side module to send SDN-Mon messages to a specific switch. These messages include the Monitoring Statistics Request Message, Set Monitoring Match Fields Message, Set Sampling Ratio Message, Add Monitoring Entry Message, and Remove Monitoring Entry Message.

setMonitoringModeOn (Monitoring Match Fields, Counter Buffer Size, Query Time Interval, Switch ID): This function turns on the monitoring mode on a switch. It initiates the SDN-Mon switch-side module with the specified monitoring match fields, counter buffer size S_b, and query time interval ΔT_q. The query time interval ΔT_q can be flexibly adjusted at a later time based on the controller's requirements, by using the setQueryTimeInterval function as described below.

setMonitoringModeOff (Switch ID): This function turns off the monitoring mode on a switch. All data structures of SDN-Mon switch-side module will be cleaned up accordingly.

addMonitoringEntry (Monitoring Match Fields Pattern, Switch ID): This function adds a new monitoring entry into the switch-side monitoring table. For monitoring a specific flow, the controller uses this function to insert a new monitoring entry into the switch-side monitoring table for monitoring. The controller-side module encapsulates the monitoring match fields pattern into an Add Monitoring Entry Message and sends it to the switch specified by its switch ID.

removeMonitoringEntry (Monitoring Match Fields Pattern, Switch ID): This function removes an existing monitoring entry from the switch-side monitoring table, when the controller determines not to keep monitoring a specific flow/entry. The switch processes this command by removing the corresponding monitoring entry from the monitoring table and by concurrently creating a new Bloom filter element corresponding to the entry to add into the Bloom filter for marking the flow as non-monitoring.

resetMonitoringTable (Monitoring Match Fields, Counter Buffer Zize, Query Time Interval, Switch ID): This function cleans up the existing monitoring table and resets it with the specified monitoring match fields, counter buffer size S_b, and query time interval ΔT_q.

setSamplingRatio (Sampling Ratio Value, Switch ID): This function sets a sampling ratio for the switch-side monitoring database. It can be reused subsequently to adjust the sampling ratio and control the monitoring table size to avoid the overflow or overload problem.

setQueryTimeInterval (Query Time Interval, Switch ID): This function sets a new query time interval ΔT_q. The controller uses this function to notify the monitoring switch each time it changes the query time interval. The switch then calculates and updates the counter-buffer-update time interval ΔT_u based on this new query time interval. The counter buffers on the switch will be updated based on the new value of ΔT_u accordingly.

setOverflowNotificationThreshold (Threshold Value, Switch ID): This function supports setting a threshold for the

monitoring switch to pre-alert the controller concerning the overflow of the monitoring table. When the number of existing monitoring entries exceeds this threshold, which can be set based on the switch capacity, the switch will send an Overflow Notification Message to its controller to notify the controller. The ontroller can reuse this function at subsequent times to adjust the threshold to avoid the overflow problem.

2.4 SDN-Mon Messages

The SDN-Mon messages are designed to allow the SDN-Mon controller-side and switch-side modules to communicate with each other to process SDN-Mon functionalities. We leverage the Experimenter Message Type of the OpenFlow protocol [12] in defining SDN-Mon messages. This allows SDN-Mon to use the existing OpenFlow protocol for its controller-side module and switch-side module communication, which makes it easy for deployment and avoiding any possible conflict or incompatibility in the communication channel. Experimenter extensions provide a standard way for OpenFlow switches to implement additional functionality within the OpenFlow message type space. This is a staging area for features meant for future OpenFlow revisions. A typical experimenter message is composed of an experimenter ID, an experimenter type, and experimenter arbitrary data. In SDN-Mon messages, the experimenter ID is set to a unique 32-bit constant, which allows the SDN-Mon modules to differentiate between SDN-Mon messages and other experimenter messages that may exist in the communication channel. The Experimenter Type field in SDN-Mon is set with the 32-bit ID of the module sending the message (note that both of the SDN-Mon switch-side and controller-side modules are assigned with a unique ID for management purposes). With these settings, each SDN-Mon message includes: an OpenFlow header, a SDN-Mon experimenter ID (32-bit), a SDN-Mon module ID (32-bit), and a message content.

3. Implementation

Our implementation of SDN-Mon consists of the switch-side and controller-side modules. The current implementation of the switch-side module works on a Lagopus switch [9] and the controller-side module works on a Ryu controller [17]. The Lagopus switch is a software OpenFlow switch that exhibits a high performance, is easy to deploy, and has been used widely in the research community recently. The switch-side module is implemented using C programming language and the controller-side module is implemented using Python. In our implementation, only about 1,500 lines of C code is added into the Lagopus switch for the SDN-Mon switch-side module and only a few hundred lines of Python code is added to the Ryu controller for the SDN-Mon controller-side module, which requires only a limited amount of processing power for the SDN-Mon monitoring mechanism. Although SDN-Mon extends switches, it remains deployable since it runs on the general-purpose processors of the switches.

4. Evaluation

4.1 Experiment Environment

We conducted experiments with a real SDN network as illustrated in **Fig. 4**. This network is composed of an SDN software switch, a controller and two hosts connecting to the switch (a sender and a receiver). The software switch is a Lagopus software switch version v0.2.0 with the SDN-Mon switch-side module, and the controller is the Ryu controller version v3.26 with the SDN-Mon controller-sided module. We ran a SDN-Mon supported Lagopus switch and a default/unmodified Lagopus switch to evaluate the performance overhead.

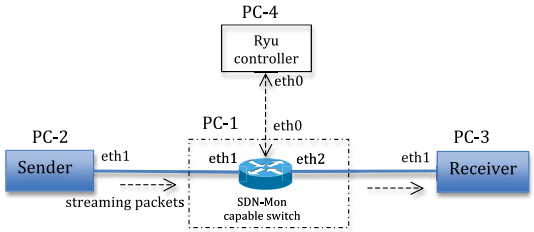

Fig. 4 Experimental setup.

Table 1 Hardware configuration of experimental network.

	CPU	Memory size	NIC type
PC-1 (SDN-Mon switch)	Intel Xeon E5-1603 2.8 GHz (4 cores)	8 GB	Intel 10-Gb X540-AT2 (eth1, eth2), Intel I217-LM (eth0)
PC-2 (Sender)	Intel core i7 3.4 GHz	8 GB	Intel 10-Gb X540-AT2
PC-3 (Receiver)	Intel Xeon CPU W3530 2.80 GHz	12 GB	Intel 10-Gb X540-AT2
PC-4 (Ryu controller)	Intel core 2 Duo 2.66 GHz (2 cores)	3 GB	Intel 82562V-2 10/100

The hardware configuration of the experimental network is summarized in **Table 1**. The software switches in our experiments run with DPDK v2.2.0 [6] for increasing the speed of packet processing. The switch in each experiment worked on the server PC-1, a physical computer with a 2.8-GHz CPU (4 cores with 256 KB of L2 cache, 10 MB of L3 cache) and 8-GB RAM. The Ryu controller worked on a separate computer (PC-4) and connected to a switch through a LAN. Two Intel 10-Gigabit X540-AT2 NICs were installed in PC-1 to handle switch ports (represented by interfaces 'eth1' and 'eth2' of the switch, as shown in Fig. 4). An Intel 10-Gigabit X540-AT2 NIC was also installed in each host, which was connected to the corresponding NIC of the switch.

We ran the SDN-Mon-supported Lagopus switch and the Ryu controller for the experiments. We set monitoring match fields to 5 tuples consisting of a source IP address, source port, destination IP address, destination port, and protocol, which are popular matching fields used in a variety of monitoring applications. The S_b was set to 5 and the ΔT_q was set to 10 seconds, which results in packet and byte counters being updated in every 2 seconds. We injected a stream of packets from host 1 to host 2. We measured the throughput, then repeated the same experiments for the default Lagopus switch to evaluate the performance overhead of the SDN-Mon modules. We used bmon [2] for throughput measurement at both the sender and the receiver sides to measure the exact packet-injecting rate at the sending host and packet-receiving rate at the receiving host. For injecting packets, we used the tcpreplay tool [23], which supports the high-speed injection of packets.

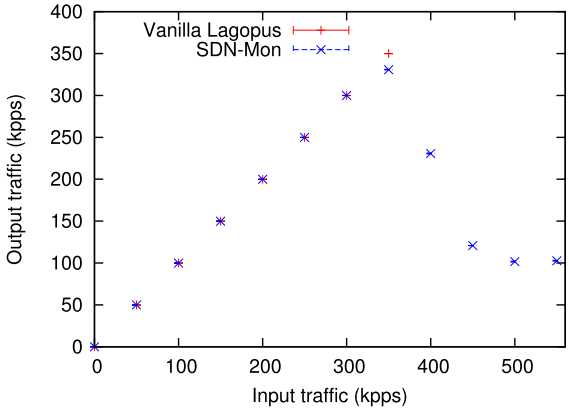

Fig. 5 Overhead of SDN-Mon at various traffic input rate.

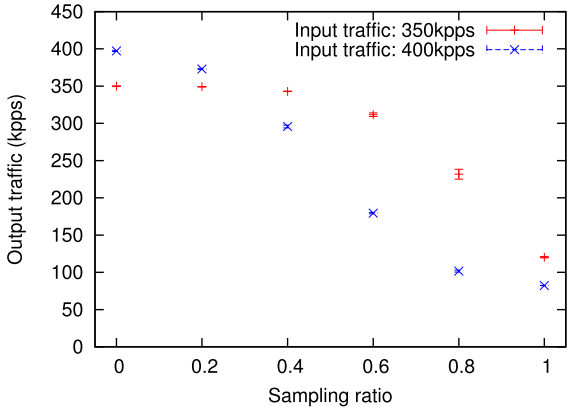

Fig. 6 Impact of sampling ratio on the overhead of SDN-Mon-supported switch.

4.2 SDN-Mon Switch Overhead Evaluation

We evaluated the performance of the SDN-Mon-supported Lagopus switch based on two aspects: the overhead of the SDN-Mon-supported Lagopus switch compared to the original Lagopus switch, and the impact of the sampling mechanism on the scalability of SDN-Mon. The evaluations were conducted using a pcap trace, which consists of 254,022 packets with a fixed packet size of 64 bytes and was looped 10 times in each experiment. In the experiments for SDN-Mon overhead, we set the sampling ratio of SDN-Mon to 0.5 and measured the packet output rates in kpps through different packet-injecting rates from 0 to 400 kpps. We measured the sending/input rates R_is at the network interface eth1 of the sender, as shown in Fig. 4, which is connected to the network interface eth1 (receiving interface) of the SDN-Mon-supported Lagopus switch. The receiving/output rates R_os were measured at network interface eth1 of the receiver, which is connected to the network interface eth2 (sending interface) of the switch. Each experiment was repeated 5 times and the sample means and standard deviations of the measured values were calculated for the final results.

Figure 5 shows the R_os of SDN-Mon-supported and vanilla/default Lagopus switches. The experimental results show that with those experimental setups, SDN-Mon performed ideally at R_i from 0 to 300 kpps, with the ratio of R_o to R_i (RoT) almost 100% and without any packet loss. The RoT started to decrease when the input traffic rate increased over 300 kpps. The SDN-Mon-supported Lagopus switch still resulted in a minor performance overhead for R_i from 300 kpps to 350 kpps.

Although the RoT decreased rather quickly from R_i of 350 kpps, it is still reasonable because we injected a large number of monitoring entries for a worst case evaluation. Every switch has a certain limit of memory and processing capacity to handle a certain threshold of packet injecting speed. The pcap file we used has about 180,000 5-tuple-based flows, which results in about 9,000 monitoring entries created in the monitoring table (since the sampling ratio was 0.5). This is a large number of monitoring entries compared to only a few tens of flow entries created in the flow tables of the orginal Lagopus switch. This results in a rather large difference in the performance of the SDN-Mon-supported Lagopus switch compared to the original Lagopus switch from R_i of 350 kpps. Even though our implementation of the monitoring table is based on the hash table data structure, which processes faster than the linear search table as in the current Lagopus implementation, and the SDN-Mon Bloom filter processing is lightweight. In practical cases, the monitoring functionality of a controller application may require a smaller number of monitoring entries, which can be handled properly by the memory and processing capability of a normal switch, than that in our experiments. In this case, the controller application can balance between the number of flow entries and the number of monitoring entries (e.g., it can set general rules with wildcards to control the number of stored flow entries) for efficiently handling packet forwarding and SDN-Mon-based fine-grained monitoring.

We also conducted experiments to evaluate the impact of the SDN-Mon sampling mechanism. We injected packets at R_i of 350 kpps and 400 kpps and measured the output traffic. We set different sampling ratios in SDN-Mon from 0 to 1 to observe the effect of the sampling ratio on the overhead of the SDN-Mon-supported Lagopus switch. We repeated each experiment 5 times and calculated the sample means and standard deviations of the measured values for the final results. **Figure 6** shows the positive effect of the sampling mechanism on the scalability of SDN-Mon. The sampling ratio of $K\%$ represents the case in which $K\%$ of the total number of packets is processed at the monitoring table of SDN-Mon, and the rest are processed at the Bloom filter. The experimental results show that when the sampling ratio decreased, the output traffic rate increased correspondingly. Moreover, the impact of the SDN-Mon's sampling mechanism on the system scalability is more significant for higher packet-injecting rates (400 kpps in Fig. 6). This shows that the efficient sampling mechanism of SDN-Mon is beneficial regarding scalability to be able to adapt to various network scales.

4.3 SDN-Mon System Overhead Evaluation

We evaluated the system overhead of SDN-Mon from two aspects: (1) the system overhead in various monitoring-table sizes (the number of active monitoring entries in the monitoring table) without background traffic, and (2) the system overhead at various R_i of background traffic. To evaluate the system overhead of SDN-Mon, we built a controller program that leverages the SDN-Mon monitoring APIs to query monitoring statistics in the SDN-Mon-supported Lagopus switch from the controller. The system overhead is represented by the elapsed time of a round

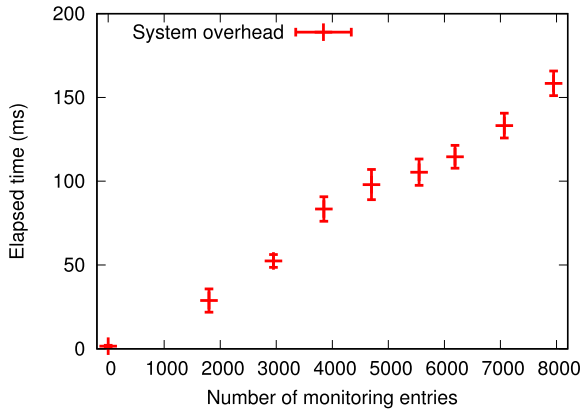

Fig. 7 System overhead at various SDN-Mon monitoring-table sizes.

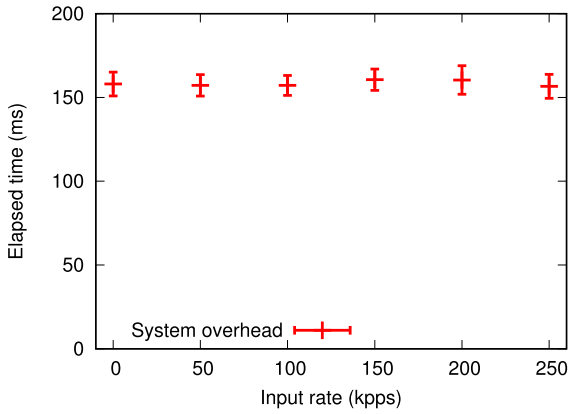

Fig. 8 System overhead at various packet input rates.

trip since the controller sends a monitoring-statistics request to the switch until it completes receiving all statistical data from the switch. This elapsed time consists of the time for the monitoring-statistics request to reach the controller since it was sent, the time for processing in the switch to collect the requested data and create a monitoring statistics reply message, and the time since the reply message was sent until it reaches the controller. In this experiment, the sampling ratio of SDN-Mon was set to 0.5. The controller program in our experiment sent a monitoring-statistics request to the switch every 10 seconds periodically, and the timestamps were marked to observe the exact time when the controller sent the request and when it received reply data from the switch.

For the performance evaluation (1), we used pcap traces containing different numbers of 5-tuple-based flows from 0 to 15,900, which resulted in 0 to 7,950 5-tuple entries in the monitoring table of the SDN-Mon supported Lagopus switch. In each experiment, we injected the packets in the pcap file and started to observe the system elapsed time when the sending host completed injecting the packets. We observed the system elapsed time 5 times and calculated the sample means and standard deviations. **Figure 7** shows the observed system eplapsed time. We observed that the elapsed time increased from 1.6 to 158.4 ms when the number of entries in the monitoring table increased from 0 to 7,950. This is reasonable because for a larger number of monitoring entries, the switch consumes more time for collecting the statistics from all entries, and transfering a larger amount of statistical data also results in a greater delay time in the communication channel between the switch and controller. The experimental results show that the system elapsed time of SDN-Mon-based monitoring is very small, even for pulling statistical data of thousands of monitoring entries.

We conducted the performance evaluation (2) by using a pcap trace containing 15,900 5-tuple-based flows (205,000 packets), which creates 7,950 5-tuples entries in the monitoring table of the SDN-Mon-supported Lagopus switch accordingly at a sampling ratio of 0.5. We injected the packets at varios R_i from 0 to 250 kpps to create the background traffic in the experiments. We observed the elapsed time of the system 5 times and calculated the sample means and standard deviations. **Figure 8** shows the elapsed time of the monitoring system in different R_i from 0 to 250 kpps. The experimental results show that the overhead of background traffic was negligible compared with the cases of no background traffic.

5. Related Work

Existing approaches proposed monitoring platforms or mechanisms to support the requirements of monitoring-based applications in SDN (e.g., anomaly detection, QoS, network link utilization, throughput measurement). The most closely related approach to ours is UMON [26], which proposes a mechanism for traffic monitoring in Open vSwitch [13]. UMON is aimed at providing flexible and fine-grained monitoring by supporting traffic monitoring based on non-routing fields, subflow monitoring, and decoupling monitoring from the forwarding functionality in the switches. Although the idea of UMON may benefit for more fine-grained traffic monitoring in SDN networks, it does not provide an architectural design. Thus, the current design mainly fit for only Open vSwitch. Moreover, UMON does not support any sampling mechanism, which can result in a lack of scalability for monitoring, especially when the number of flows that switches require to process in reality can be easily in the millions, larger than the supported processing capability of current SDN hardware switches.

Other studies proposed changes to the SDN data plane to improve the performance of SDN monitoring. Avant-Guard [19] outlines the challenge of the inherent communication bottleneck that arises between the data and control planes in SDN. It extends the OpenFlow data plane to reduce the amount of switch-controller interactions and delays in the control channel. OFX [20] proposes an OpenFlow extension similar to Avant-Guard, but simplifies the deployment since it runs on unmodified hardware switches. DevoFlow [5] proposes triggers and reports, and approximate counter mechanisms for reducing the amount of switch-controller interactions. These approaches basically rely on existing flow tables in switches for monitoring and do not support any efficient sampling mechanism. Thus, they are not well-supported for fine-grained monitoring and lack flexibility and scalability, especially for networks that process a large number of flows.

Toward reducing the controller-switch communication overhead in SDN-based monitoring, other approaches support adaptive polling-switches selection mechanisms and flow-statistics aggregation mechanisms to reduce the communication cost of monitoring globally [4], [21]. Others propose a similar switches

selection scheme for querying flow statistics for estimating traffic matrix with a low overhead [24], or prediction based flow counting mechanism that dynamically changes the aggregation of monitoring flows to balance the monitoring overhead and the anomaly detection accuracy [29]. These proposals differ from ours since their idea is basically distributing and balancing the load or overhead over switches to reduce the overall overhead of the network, and their monitoring functionality mainly relies on unmodified flow tables in switches, which remain inflexible and unscalable in handling a large number of flows in networks. However, these distributed approaches can be leveraged in deploying SDN-Mon framework for monitoring over multiple switches.

Other approaches use existing solutions (e.g., sFlow [15]) in handling the monitoring functionality in SDN networks instead of using SDN switches [22], [28]. These studies present sampling-based approaches that use the sFlow agent at switches to capture packet-header samples from the network. The drawbacks of these approaches are that they strictly require additional hardware-device deployment for the flow-collecting system, and are not integrated with the OpenFlow platform. Thus, they may not be applicable in some practical cases. Moreover, they add more possible delay and overhead in the processing of a network since every packet-header sample needs to be transmitted to an external device.

Other related studies support accurate monitoring of per-flow throughput, packet loss, and delay metrics [25], or uncovering forwarding problems due to hardware or software failures in SDN switches [14]. Other approaches have been proposed for measuring link utilization by capturing and analyzing control messages between switches and the controller [27], or dynamic provisioning of network resources with guaranteed QoS upon changes in the application requirements [3]. These works are mainly for measuring quality of service and reliability in SDN, and basically differ from our approach since they are not aimed at providing a monitoring framework that is beneficial for various monitoring purposes of controller applications.

6. Discussion

For scalability of the monitoring functionality, SDN-Mon supports an efficient sampling-based mechanism to control the processing and memory overhead in network switches. The framework allows the controller to control and limit the total number of monitoring entries to avoid the monitoring-related overflow problem in networks. With the controller-side monitoring APIs, the controller can easily check the total number of existing monitoring entries in the monitoring table to avoid the overflow issue in the table. When the number of monitoring entries reaches a certain threshold, which is set by the controller based on the capability of the switch, the controller can increase the current sampling ratio to limit or control the increase in the number of monitoring entries in the monitoring database.

The SDN-Mon framework is designed with a high priority for minor overhead in the processing of switches and for lowering the frequency of monitoring-based controller-switch communication, while providing flexible monitoring functionality for controller applications. With the support of a Bloom filter for marking ignored or non-monitoring flows, switches query the controller only once for every new flow. If the sampling mechanism ignores a new incoming flow from monitoring, a new Bloom filter element, which corresponds to the flow, will be added to the Bloom filter to mark that flow as 'non-monitoring'. The following incoming packets of the same flow will be checked with the Bloom filter first and then be ignored since this flow is marked in the Bloom filter by its first incoming packet. This mechanism prevents the switches from conducting table lookups for the following packets of non-monitoring flows and from sending improper packet-in messages to query the controller for processing these packets. This saves the processing power for the switch and annuls the overhead in the controller-switches communication channel caused by improper queries by the following incoming packets of non-monitoring flows. We will consider supporting reliably removing exisiting bloom filter elements with the Cuckoo filter [7], in the next version of our implementation. This supports the case in which controller applications want to add monitoring entries that were decided as non-monitoring to the monitoring table.

The processing and memory overheads of the additional data structures in a SDN-Mon-supported switch can be balanced because the SDN-Mon switch-side module can be used to handle the monitoring functionality, allowing the control plane to focus on only the forwarding functionality. With this support, the control plane can determine forwarding rules flexibly to balance the cost of its forwarding functionality and SDN-Mon-supported monitoring functionality. For instance, it can use more general forwarding rules with wildcards inserted into switches for forwarding. This results in maintaining proper forwarding functionality of the switch, while leaving more processing and memory for the SDN-Mon switch-side module to process its fine-grained monitoring functionality. Moreover, the Bloom filter used in SDN-Mon is scalable since it is a lightweight data structure that consumes very little processing and memory power in the switch. Therefore, the trade-off between the performance and the benefits of SDN-Mon for monitoring is reasonable. The SDN-Mon framework supports both TCP-based and non-TCP flow monitoring, which would benefit the majority of network systems and applications.

7. Conclusion and Future Work

We proposed SDN-Mon, a new framework for fine-grained efficient traffic monitoring in SDN. This framework separates monitoring functionality from forwarding to support flexible monitoring to serve a variety of monitoring-based applications. With SDN-Mon support for monitoring, the control plane can determine more proper forwarding rules to handle the forwarding functionality of the switches to avoid overflow in the switches' flow tables in balance with the cost of overhead for monitoring in the switches. The monitoring modules in SDN-Mon are also scalable since its efficient sampling-based monitoring mechanism helps prevent switches from frequent queries to the control plane, which may cause unnecessary overhead in the controller-switch communication channel and limit the number of monitoring entries to adapt with the supported capacity of switches. We implemented SDN-Mon on the Lagopus switch and evaluated its performance to show that its overhead is acceptable while providing

more efficient fine-grained and flexible monitoring in SDN networks. In the future, we plan to leverage SDN-Mon for developing various monitoring-based solutions for network management in SDN.

References

[1] Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors, *Comm. ACM*, Vol.13, No.7, pp.422–426 (1970).
[2] Bmon, available from ⟨https://github.com/tgraf/bmon⟩.
[3] Bueno, I., Aznar, J.I., Escalona, E., Ferrer, J. and Antoni Garcia-Espin, J.: An opennaas based sdn framework for dynamic qos control, *IEEE SDN4FNS'13*, pp.1–7 (2013).
[4] Chowdhury, S.R., Bari, M.F., Ahmed, R. and Boutaba, R.: Payless: A low cost network monitoring framework for software defined networks, *IEEE NOMS'14*, pp.1–9 (2014).
[5] Curtis, A.R., Mogul, J.C., Tourrilhes, J., Yalagandula, P., Sharma, P. and Banerjee, S.: DevoFlow: Scaling flow management for high-performance networks, *ACM SIGCOMM Computer Communication Review*, Vol.41, No.4, pp.254–265, ACM (2011).
[6] DPDK: Data plane development kit, available from ⟨http://dpdk.org/⟩.
[7] Fan, B., Andersen, D.G., Kaminsky, M. and Mitzenmacher, M.D.: Cuckoo filter: Practically better than bloom, *ACM CoNEXT'14*, pp.75–88 (2014).
[8] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., et al.: B4: Experience with a globally-deployed software defined WAN, *ACM SIGCOMM Computer Communication Review*, Vol.43, No.4, pp.3–14, ACM (2013).
[9] Lagopus switch: A high performance software OpenFlow 1.3 switch, available from ⟨http://lagopus.github.io/⟩.
[10] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J.: OpenFlow: Enabling innovation in campus networks, *ACM SIGCOMM Computer Communication Review*, Vol.38, No.2, pp.69–74 (2008).
[11] Open Networking Foundation: Software Defined Networking: The new norm for networks, *White paper* (2012).
[12] Open Networking Foundation: OpenFlow Switch Specification verion 1.5.0 (2014).
[13] Open vSwitch, available from ⟨http://www.openvswitch.org/⟩.
[14] Peresini, P., Kuzniar, M. and Kostic, D.: Monocle: Dynamic, Fine-Grained Data Plane Monitoring, *ACM CoNEXT'15*, pp.1–13 (2015).
[15] Phaal, P., Panchen, S. and McKee, N.: InMon Corporation's sFlow: A Method for Motoring Traffic in Switched and Routed Networks, RFC3176 (2001).
[16] POX Controller, available from ⟨http://www.noxrepo.org/pox/about-pox/⟩.
[17] Ryu SDN Framework, available from ⟨http://osrg.github.io/ryu/⟩.
[18] Sezer, S., Scott-Hayward, S., Chouhan, P.-K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M. and Rao, N.: Are we ready for SDN? Implementation challenges for software-defined networks, *IEEE Communications Magazine*, Vol.51, No.7, pp.36–43 (2013).
[19] Shin, S., Yegneswaran, V., Porras, P. and Gu, G.: AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks, *ACM CCS'13*, pp.413–424 (2013).
[20] Sonchack, J., Aviv, A.J., Keller, E. and Smith, J.M.: Enabling Practical Software-defined Networking Security Applications with OFX, *NDSS'16*, pp.1–15 (2016).
[21] Su, Z., Wang, T., Xia, Y. and Hamdi, M.: FlowCover: Low-cost flow monitoring scheme in software defined networks, *IEEE GLOBECOM'14*, pp.1956–1961 (2014).
[22] Suh, J., Kwon, T.T., Dixon, C., Felter, W. and Carter, J.: OpenSample: A low-latency, sampling-based measurement platform for commodity SDN, *IEEE ICDCS'14*, pp.228–237 (2014).
[23] Tcpreplay, available from ⟨http://tcpreplay.synfin.net/wiki/tcpreplay⟩.
[24] Tootoonchian, A., Ghobadi, M. and Ganjali, Y.: OpenTM: Traffic matrix estimator for OpenFlow networks, *PAM'10*, Springer, pp.201–210 (2010).
[25] Van Adrichem, N.L., Doerr, C. and Kuipers, F.A.: OpenNetMon: Network monitoring in openflow software-defined networks, *IEEE NOMS'14*, pp.1–8 (2014).
[26] Wang, A., Guo, Y., Hao, F., Lakshman, T. and Chen, S.: UMON: Flexible and Fine Grained Traffic Monitoring in Open vSwitch, *ACM CoNEXT'15*, pp.1–7 (2015).
[27] Yu, C., Lumezanu, C., Zhang, Y., Singh, V., Jiang, G. and Madhyastha, H.V.: Flowsense: Monitoring network utilization with zero measurement cost, *PAM'13*, Springer, pp.31–41 (2013).
[28] Zaalouk, A., Khondoker, R., Marx, R. and Bayarou, K.: OrchSec: An orchestrator-based architecture for enhancing network-security using Network Monitoring and SDN Control functions, *IEEE NOMS'14*, pp.1–9 (2014).
[29] Zhang, Y.: An adaptive flow counting method for anomaly detection in SDN, *ACM CoNEXT'13*, pp.25–30 (2013).

Xuan Thien Phan is a Ph.D. candidate at the Graduate University for Advanced Studies (SOKENDAI). He received his Master degree in computer science from HoChiMinh City University of Technology in 2013. His current research interests are software-defined networks, Internet traffic measurement, and network anomaly detection.

Kensuke Fukuda is an associate professor at the National Institute of Informatics (NII). He earned his Ph.D degree in computer science from Keio University in 1999. He worked in NTT laboratories from 1999 to 2005, and joined NII in 2006. He was a visiting scholar at Boston University in 2002 and a visiting scholar at the University of Southern California/Information Sciences Institute in 2014–2015. He was also a researcher of PRESTO JST (Sakigake) in 2008–2012. His current research interests are Internet traffic measurement and analysis, intelligent network control architectures, and the scientific aspects of networks.