
Pang-Wei Tsai, Chun-Yu Hsu, Mon-Yen Luo and Chu-Sing Yang
Department of Computer Science & Information Engineering, National Kaohsiung University of Applied Sciences / No.415, Chien Kung Rd., Sanmin District, Kaohsiung City, Taiwan.
(e-mail: myluo@kuas.edu.tw)

Abstract—The measurement is an important issue in network management. The measurement results are able to provide helpful information for network administrators. Operating issues such as traffic engineering and traffic accounting also rely on the basis of it. However, the trade-off between accuracy and overhead has to be under consideration. Nowadays, though the Software-Defined Networking technology has been widely changed and made new approaches on the network architectures, monitoring the traffic is still a crucial task. This paper presents a group-based flow monitoring scheme to adaptively measure flows in the SDN-enabled network. According to the designed algorithms, the sampling frequency is adjusted from time to time. By doing this, the measurement becomes more adaptive and efficient. This paper also presents a demonstration of the design. The paper aims to illustrate the idea and to share the experience of designing an alternative way for flow measurement in the SDN-enabled network.

Index Terms—SDN, OpenFlow, Traffic Measurement, Monitor, Polling, Group-based, Statistic

I. INTRODUCTION

For network operators, the measurement is one of the most important issues. The measurement results not only perform the status of network but provide consulting for management. The measured data is also able to help network operators monitoring their networks and dealing with the operations such as traffic engineering and traffic-anomaly detection.

Since the scale of the global network has continued to grow, its scope has become increasingly complicated. Therefore, many researchers are developing new network architectures to meet the challenges. The Software-Defined Networking (SDN) is one of the innovations proposed to be the achievement. The SDN provides flexible network control policies compared to traditional network components. It separates control plane and data plane, which supports flow-aware configurations and rule-driven actions to forward packets. The OpenFlow [1] is one of the SDN standards for achieving the programmability of network as just mentioned. The OpenFlow standardizes the communication between the controller and the switch. NOX [2] is the classic design of controller for managing the OpenFlow network. Several studies have reviewed the SDN as well as the OpenFlow [3-6], and many SDN technologies are widely adopted in network researches currently.

Comparing the measurement technique between traditional network and OpenFlow network, many classical problems such as data organization and presentation are almost resemble. However, the advantage of the OpenFlow is that its centralized control plane provides the global view of entire network. This distinguishing functionality allows network operators track each flow in the entire network. The evaluation leads the challenge in network measurement. There are many researchers use opened and flexible ways [7] to design measurement methods for gaining efficiency in different operating scenarios.

When deploying the OpenFlow switches as edge switches in the local area network on campus1 for experimental deployment, authors of this paper have met the requirement on monitoring traffic. The original measurement methods for traditional switches are using SNMP [8] and RRDTools [9] (e.g., cacti [10]) to export MRTG [11] charts to observe per-port traffic. For auditing traffic in detail, the logs exported by NetFlow [12] are used to analyze the flows. However, The OpenFlow switches2 deployed in authors’ environment have not native support to export the traffic logs (e.g., NetFlow and sFlow [13]) when operating in OpenFlow mode. For more, the original per-port statistics can not observe the per-flow traffic in SDN environment. Authors have to implement alternative methods for data collection, organization and processing to measure flow stats under this circumstance.

1 A campus network is operated by the authors, https://noc.ee.ncku.edu.tw
2 The solutions used in experimental deployment are OpenvSwitch, NetFPGA and Cisco Catalyst 3K series with beta firmware. The new OpenFlow plug-in software of Cisco Catalyst 4500 series is just released, and it will be evaluated in the future plan.
To develop a satisfied method for measuring per-flow traffic on the edge side of the network, this paper presented a prototype design for measurement in the OpenFlow network. The research focuses on developing an adaptive method to monitor the flows. The remainder of this paper is organized as follows. Background and related work are introduced in Section 2. The considerations of the design are discussed in Section 3. The system design is presented in Section 4. The experiment and verification are described in Section 5. The conclusions and future work are summarized in Section 6.

II. BACKGROUND AND RELATED WORK

This section introduces the background and related work. The first sub-section presents the terms and notions for getting the flow stats by the OpenFlow protocol briefly. The second sub-section introduces several proposed methods for monitoring traffic in the OpenFlow network.

A. Basic terms and notions

To get the flow stats from counters on the OpenFlow switch, there are two ways: polling data by the controller or pushing data by the switch. The former way is using the controller to send the query message to the OpenFlow switch, and the controller polls the information from the switch to the controller. There is a message format for querying flow stats in the OpenFlow protocol. The polling procedure is shown as Figure 1(a). The latter way is through the default action of processing expired flow rules. When a flow rule is expired, the OpenFlow switch not only deletes this rule but sends FlowRemoved message to inform the controller. This message includes the counter values related to this rule, and the controller gets the stats of the flow after it receives the FlowRemoved message. The procedure of stats collection is shown as Figure 1(b).

According to the OpenFlow specification 1.3.0 (wire protocol 0x04 [14]), both polling and pushing methods mentioned above have the same data structure for carrying the values of counters. There are three values in this structure: duration, packet_count and byte_count. The duration represents alive time of this flow; the packet_count shows the accumulated number of forwarded packets; the byte_count summarizes the total volume of forwarded packets.

B. Related work for measurement in SDN

For dealing with the measurement issues in the SDN, several methods are proposed [7]. Most of methods can be simply classified into several types: push, pull (query) and span (mirror). Followings are three typical designs which are representative developments.

1) FlowSense: For reducing the overhead of measurement, many researches try using efficient ways to query the data. In contrast, the FlowSense [15] uses completely passive-way to make zero measurement cost. The design of the FlowSense sends no query message from the controller while the stats of each flow are still able to be collected. According to OpenFlow protocol, flow information (i.e., duration, packet_count and byte_count) are included in the FlowRemoved message, which will be pushed by switch automatically when a flow rule is expired. The FlowSense uses this way to collect flow stats. There are two major modules in the FlowSense: control traffic parser and utilization monitor. The parser checks two control messages in the control plane: PacketIn and FlowRemoved. The monitor module collects the information to present traffic statistics. This design achieves a cost-less method to measure the flows, however, the statistics of each flow can only be calculated when the flow rule is expired.

2) PayLess: PayLess [16] is a pull-based monitoring framework built in SDN controllers. There are four major developments in the PayLess: Request Interpreter, Scheduler, Switch Selector and Aggregator & Data Store module. A formula in PayLess compares the last traffic volume (i.e., byte_count) and current one, coordinating the polling rate by degrees if the ratio is over the upper- or lower-threshold. If so, the next polling action is going to move to the neighborhood stages with different interval time. The authors of PayLess evaluated their development in several experiments, and the development aims to have better measurement on actual utilization in experiments. The PayLess is a good example for pull-based method. However, it is important for authors to make adaptive polling actions since the flows are unconscious under the circumstance. Changing the polling frequency to the suitable rate directly is an essential purpose in the design.
3) OpenSAFE: OpenSAFE [17] is an extensible and scalable method for mirroring and measuring large amount traffic on a span system. Authors of OpenSAFE take a real environment as an example to illustrate the challenges of monitoring span traffic. OpenSAFE consists of three components: abstractions for processing traffic, a policy language to control the measurement and a modified controller to manage the programmed network. The evaluation of OpenSAFE shows that the implementation undertakes the traffic of referenced network for monitoring and scaling the measurement successfully. The evaluation environment of OpenSAFE is in the campus network (which is the same as authors’ environment), while it is suitable to implement at the core network rather than the edge side.

III. DISCUSSION
After observing the traffic statistics of traditional switches, authors notice that the traffic through the edge switch (which is nearing the end-hosts) is often busy and unconscious. For example, when the application with highly networking dependency are activated (e.g., FTP client and live broadcast player) on the end-hosts, the traffic amount is increasing in a short time. In contrast, some other applications (e.g., e-mail editor) have less traffic utilization. The MRTG chart in Figure 2 shows this circumstance. When replace traditional switches with OpenFlow switches in the edge side, authors need to figure out a way to acquire the information of flows efficiently.

Furthermore, for measuring the network traffic, classically, there are two means [18]: active and passive. Both polling and pushing methods described in last section belong to passive measurement. Although the controller is able to send customized probe packets for active measurement, this method is usually used in inactive detection (e.g., latency measurement and topology discovery). It is out of the scope of this paper. The design in this paper is concerning about developing an alternative methods in the OpenFlow way for observing traffic of flows.

IV. SYSTEM DESIGN
In following segments, this section gives an overview of proposed method and introduces a group-based flow monitoring scheme for measuring the traffic.

A. System architecture
In traffic statistics, there are many developing issues [19] such as collection, organization, analysis, interpretation and presentation in data processing stages. The proposed method in this paper is concerning about the collection, organization and presentation. Due to the design concept of OpenFlow, the controller has the granted role to access all the switches. The design motivation in this paper is to use this characteristic, trying gathering the flow stats efficiently and reducing the overhead of the control plane. The designed architecture is shown in Figure 3.

There are four main components in this design, all of them are able to be constructed as modules within the controller. A database is used to store processed data among these components. Details of designed components are described below:

1) Flow Status Tracker: Observing the flow status is an important functionality in the design. For new added flows, the traffic stats need to be preserved; for expired flows, the collecting actions should be terminated. To achieve this, this component is designed to keep tracking flow-modified messages in control plane. It maintains a table to identify active flows. Once a flow entry is added or deleted, the table will be updated in time.

2) Polling Manager: This component is designed to manage the data collection. It decides the polling frequency for each flow in two modes: pro-active and pre-active. For pro-active mode, the polling frequency of flow is appointed by the network operator. For pre-active mode, the polling frequency
Algorithm 1 FlowStatusTracker (Event $E$)

1: globals:
2: Group //Classify flows with the specific polling frequency
3: List //Store group-adjust decision for the flow
4: StatsTable //Store historical stats of active flows
5: Event //The control message about the flows
6: $f$ //Flow
7: while Event $E$ of Flow $f$ is detected do
8:     switch Event $E$ do
9:         case ?flowAdd
10:             Sort to collect stats of $f$
11:             Add flow $f$ to List$\text{default}$
12:         case ?flowRemoved
13:             Sort to collect stats of $f$
14:             Delete stats of flow $f$ in StatsTable
15:     default
16:         Do nothing
17:     end while

Algorithm 2 FlowStatsCollector (Group $G_x$, List $L_x$)

1: globals:
2: Group //Classify flows with the specific polling frequency
3: List //Store group-adjust decision for the flow
4: $f$ //Flow
5: for flows in $L_x$ do
6:     Put the flows into $G_x$
7: end for
8: Clean up $L_x$
9: for flows in $G_x$ do
10:     Send the FlowStatsRequest message to each
11:     $f$ switch for querying $f$ counters in the fixed time period
12: end for

Algorithm 3 FlowStatsAnalyzer (Event $E$)

1: globals:
2: StatsTable //Store historical stats of active flows
3: RecordTable //Store flow stats in last time
4: Event //The control message about the flows
5: if $E$ = FlowStatsReply and $E$.flow is in RecordTable
6:     $\Delta$ bit = $E$.flow.bitCount - RecordTable.flow.bitCount
7:     $\Delta$ rate = $\Delta$ bit / StatsTable.flow.timeScale
8:     $\Delta$ pkt = $E$.flow.pktCount - RecordTable.flow.pktCount
9:     $\Delta$ rate = $\Delta$ pkt / StatsTable.flow.timeScale
10:     RecordTable.flow.bitCount = $E$.flow.bitCount
11:     RecordTable.flow.pktCount = $E$.flow.pktCount
12:     StatsTable.flow.bit_rate = $\Delta$ rate
13:     StatsTable.flow.packet_rate = $\Delta$ rate
14:     PollingManager($E$.flow
15: else
16:     RecordTable.flow.bitCount = $E$.flow.bitCount
17:     RecordTable.flow.pktCount = $E$.flow.pktCount
18: end if

Algorithm 4 PollingManager (flow $f$)

1: globals:
2: Group //Classify flows with the specific polling frequency
3: List //Store group-adjust decision for the flow
4: StatsTable //Store historical stats of active flows
5: RecordTable //Store flow stats in last time
6: $N$ //The number of all groups
7: $f$ //Flow
8: if RecordTable.$f$ is exist then
9:     $r = $StatsTable.$f$.bit_rate
10:     for $x = 1$; $x < N$; $x + +$ do
11:         if threshold$_x$.lower <= $r$ < threshold$_x$.upper then
12:             Find Group$_x$
13:             if Group$_x$ == Group$_{\text{origin}}$ then
14:                 Do nothing
15:             else
16:                 Remove $f$ from Group$_{\text{origin}}$
17:                 Add $f$ into the List$_x$
18:             end if
19:         break
20:     end if
21: end for

is dynamically changed. Authors have designed a group-based scheme for adapting polling frequency in pre-active mode, which is introduced in the latter sub-section.

3) Flow Stats Collector: The flows belong to the same group are using the same polling rate in data collection. Based on the given polling frequency of each group, this component polls stats of related flows from the OpenFlow switch. After the replies arrived, this component caches the information and stores the raw data into the tables of database.

4) Flow Stats Analyzer: This component takes responsibility to organize raw data into meaningful information for analysis and interpretation. In order to present the statistics of each flow, a web-based dashboard is planned to be used to present the real-time traffic charts of active flows.

B. Group-based flow monitoring scheme

According to the OpenFlow specification, packet_count and byte_count of each flow are logged completely. However, the interval time of the polling action turns the calculated throughput of a flow into a sampling way. The accuracy and overhead of measurement depends on the polling frequency. Frequently intervening makes results accurate but brings excessive load on control plane. On the contrary, lowing checking frequency relieves system but decreases the accuracy.

For balancing the accuracy and efficiency in the measurement, a group-based scheme is used to adjust the polling frequency. There are many group (for recognizing different polling timescales), and each active flow is going to be disposed into one of the groups. Each group has a timer to countdown its time for triggering polling action, when time dures, the Flow Stats Collector sends query messages to get stats of related flows in that group. As time goes by, flows may be dynamically moved to other groups by the recommended adjustments. The Polling Manager is designed to make the

The FlowAdd is a customized message developed by authors, which is used to represent a completed activation of the new added flow.
decisions. By doing this, the major benefit is to monitor the busy and unconscious flows precisely with frequent polling rate. Another advantage is reducing the communication cost between switch and controller but still keeping accuracy of statistics analysis to an extent.

C. Development

To realize the scheme, there are four components need to be developed. The Flow Status Tracker uses Algorithm 1 to detect the new added flows, putting them into the default group. The Flow Stats Collector runs Algorithm 2 for polling stats of each flow in the group. The returned messages are passed to Flow Stats Analyzer, making compilation in Algorithm 3. The Flow Stats Analyzer maintains two tables, RecordTable and StatsTable. The former table records the raw data of flows, while the later one stores the statistics information. In the first polling cycle, the new added flows have no historical records, and the adaptive coordination is not activated. In the second polling cycle, Flow Stats Analyzer is able to calculate statistics information of the new added flow. It analyzes traffic of flow by RecordTable and the current data in reply message. Therefore, the Polling Manager is able to make adjustment by Algorithm 4. The flows will be arranged to the group correspond to its throughput, and dynamically moves to other groups during the time. At last, when a flow is discarded, a FlowRemoved message will be send to controller. After the message is detected by the Flow Status Tracker, the polling action of this removed flow is terminated.

V. EXPERIMENT AND VERIFICATION

In order to verify the design, the developed components are implemented on an OpenFlow controller 4 (which is used Ryu [20] solution for supporting OpenFlow 1.3 protocol). The number of groups is set to 3 (this number can be increased, here it is temporary set for functional evaluation). Therefore, the flows are able to be classified into three groups with different querying intervals. The stats of each flow will be used to decide whether to change the group or not. For example, if the latest throughput of a flow falls in the area between lower and upper-threshold of another group, the system is expected to change the group of this flow subsequently. For verifying above mechanism, three experiments are proceeded.

In the first experiment, the mininet [21] simulator 5 is used to create the testing environment. There are two simulated nodes in the mininet for conducting end-hosts. In this experiment, two nodes, N1 and N2, are linked through an OpenFlow Switch, S1. The simulated OpenFlow switch in the mininet is connected to a controller with developments. The iperf [22] is used to generate the traffic for evaluation. The N1 is iperf client, and the N2 is iperf server. When N1 uses iperf to send the packets to N2, the developed components also collects the stats data from the S1. The test is set to sustain 180 seconds, and the minimum unit of time is 1 second in measurement. The output traffic of N1, input traffic of N2 and measured throughput of proposed method are shown in Figure 4. Comparing the values from these three spots, the result shows that the developed method has the approaching shape to others, and the charts only have a few deviations.

For verifying the trade-off between accuracy and efficiency, a comparison of constantly polling scheme and developed group-based polling scheme in the second experiment is shown in Figure 5. The testing environment is identical to the first experiment. For constantly polling scheme, the polling rate is setup to one second; for group-based flow monitoring scheme, the polling interval configurations are set to be $2^0$ second, $2^1$ seconds and $2^2$ seconds; the threshold values are
The web-based dashboard and presented flow statistics.

The model of controller is ASUS AS-D770 with Intel Q9400 CPU and 4GB memory. The operating system is Ubuntu 12.04 with Linux kernel 3.5.0. The model of miniNet simulator is ASUS AS-D792 with Intel Q6600 CPU and 4GB memory. The operating system is Ubuntu 12.04 with Linux kernel 3.5.0, and the miniNet version is 2.0.0. The model of developed OpenFlow Switch is a customized rack server with Intel G2020 CPU and 4GB memory. The operating system is Ubuntu 12.04 with Linux kernel 3.5.0, and the OpenSwitch software version is 2.3.1.

defined to be $10^6$ and $10^7$ bit per second (the timescales and threshold values can also be changed, here they are temporary set for functional evaluation). Figure 5 shows the statistics presentations of constantly polling scheme and proposed group-based polling scheme. Comparing to the constantly polling scheme, the proposed scheme has less accuracy while reduce the overhead by sending less queries. Although there is a gap at about 135 second due to change the polling interval time, the statistics of proposed group-based polling scheme still behaves very closely to the constantly polling scheme. The polling intervals of two schemes during the experiment are shown as Figure 6.

In the third experiment, authors replace a traditional switch with the OpenFlow switch in real environment. This switch is used to be the edge-switch in authors’ office. There are 20+ PCs connected to this switch, and 40+ con-current flows are in average. The uptime of the system is about 20 days (still continues to go). As section 4 mentioned, a web-based dashboard is designed to present con-current flows and their monitoring information. Since line chart is widely used in most MRTG-based monitoring system, this dashboard is using the Highchart [23] to present the statistics by line charts. The developed web-based dashboard is shown as Figure 7. Furthermore, the Figure 8 shows an example of detailed data of a flow in the dashboard. The horizontal axis represents the time, and vertical axis means the traffic throughput of the flow in bit per second. Figure 9 represents the adaptive polling frequency of this flow. It can be clearly seen that the polling frequency is adjusted from time to time.

VI. CONCLUSIONS AND FUTURE WORK

The measurement is a crucial task in network management, and the measured data is able to help network operators monitor their networks. For measuring flow traffic in the SDN-enabled network efficiently, this paper proposed a group-based flow monitoring scheme. This method is able to adjust the frequency of data collection by algorithms. According to the results in experiments, the proposed method adjusts the sampling frequency to reduce the overhead of the measurement, and the traffic charts of proposed method behave approaching presentation. The paper aims to share the experience of designing an alternative way for flow measurement. However, based on current work, the proposed method is just a prototype. The future research directions are planned by the authors, such as verifying the scheme with large-scale scenarios and finding the optimized parameters for algorithms.

ACKNOWLEDGEMENT

This research is financially supported by the Ministry of Science and Technology of Taiwan (under grants No.103-2221-E-006-181 / 103-2218-E-194-007) and the NARLabs (under grants No.03103A7300) for which authors are grateful.

REFERENCES
