eOpenFlow: Software Defined Sampling via a Highly Adoptable OpenFlow Extension

Guang Cheng
School of Computer Science and Engineering
Key Laboratory of Computer Network & Information Integration
(Southeast University), Ministry of Education
Southeast University, Nanjing, P.R.China 211189
Email: gcheng@njnet.edu.cn

Yongning Tang
School of Information Technology
Illinois State University
Normal, IL, USA, 61790-5150
Email: ytang@ilstu.edu

***Abstract*—Sampling is highly demanded in software defined networking (SDN) by the need to control the consumption of network measurement resources and by the need of detailed measurements from applications and service providers. OpenFlow, as the standard control protocol between SDN controller and switches, is not equipped with traffic sampling function. In this paper, we proposes a software defined sampling measurement scheme via an adoptable extension to OpenFlow called eOpenFlow. In the data plane of SDN switch, the sampling action OFPAT_OUTPUT_SAMPLING is added to sample user defined specific traffic flows. We present two different sampling rules, which are based on multi-level flow table and group-based table mechanisms, respectively. In SDN control plane, collected network samples are analyzed to realize various measurement functions. eOpenFlow has been implemented, and further evaluated via carefully designed experiments in order to verify its different sampling functions.**

I. Introduction

In Software Defined Networking (SDN), an SDN controller has a holistic view of network, and thus can orchestrate a network measurement task via delegating relevant measurement jobs to multiple coordinated switches. For many challenging tasks in traditional network measurement, such as heavy hitters (HH) and superspreaders detection, software defined measurement approach can better optimize available network resources to achieve higher measurement accuracy.

Current research on SDN network measurement focuses on utilizing the built-in counters from flow tables in OpenFlow switches. Since the limitation on the number of flow table counters, it is difficult for such an approache to meet the requirements of implementing a flexible multi-granular network measurement application.

Sampling has become an integral part of traditional passive network measurement. The incentive of adopting sampling into the new SDN paradigm is driven by the need to control the consumption of measurement resources with increasing traffic rates and the demand for detailed measurements from applications and service providers. Classical sampling methods shed light on current SDN network measurement. However, sampling in the current SDN framework (i.e., a SDN switch with north and south bound interfaces), is not naturally an easy option. Without changing the assumption on the functionalities of SDN switches and essential SDN control mechanism, a possibly feasible approach is to extend the current standard southbound interface (i.e., OpenFlow) between a SDN controller and switch. In this paper, we present a highly adoptable extended OpenFlow (i.e., *eOpenFlow*) to incorporate the sampling function into the OpenFlow framework without affecting the default network behavior controlled via OpenFlow.

In the following of this paper, we discuss the relevant background of SDN measurement in Section 2. Section 3 elaborates our design of the sampling measurement extension to OpenFlow. Section 4 shows the implementation of our approach. The performance evaluation via well-designed experiments is presented in Section 5. Finally, Section 6 concludes the paper.

II. Related Works

Currently, OpenFlow-based SDN networks rely heavily on polling flow table statistics to accomplish various traffic measurement tasks. Flowsense [1] analyzed the *PacketIn* and *FlowRemoved* information between switches and controllers to estimate the bandwidth utilization of links in OpenFlow networks. However, *FlowRemoved* messages are generated only when the traffic is timed out, and the wildcard rules cannot estimate link utilization. OpenTM [2] sent a *FlowStatisticsRequest* message with a fixed query frequency to periodically poll from OpenFlow switch for a flow table entry to infer the throughput of the link. PayLess [3] considered the query frequency with the network performance and flow changes, using a variable frequency flow statistical information collection method, through historical measurements to predict the future query frequency. CeMon [4] proposed an adaptive fine-grained polling method to optimize the flow table entry in the polling switch. OpenNetMon [5] determined the frequency of polling switches based on traffic behavioral stability. Sahri N M [6] used a higher adaptive tuning polling frequency for suspected malicious traffic. It is difficult to obtain the accurate measurement result for the sudden change of flow rate because the timely feedback of the switch can not be obtained.

Due to the limited flow table in the switch, several approaches introduced the hash sketch, programmable, sampling and other measurement methods to assist the software-defined measurement. Moshref M [7] analyzed the balance between measurement resources and accuracy under three different measurement methods, such as the flow table counter, sketch, and programmable. The proposed approach achieved a controllable precision of HH and other key flows under resource controllable conditions measuring. DevoFlow [8] designed switches with rules cloning, multi-channel support and other programmable methods to reduce the burden on the control layer, so that some traffic processing work done in switch. OpenSketch [9] designed sketch-based data flow algorithm in order to achieve the HH, Superspreader, traffic changes, traffic size distribution and other measurement tasks. OpenSample [10] used the sFlow packet sampling method in the BF switch to detect heavy hitters from the measurement samples. The Sample & Pick algorithm [11] considered the size limit of the switch's flow table space and the time-frequency limit of the insertion rule in the switch, and used the sampling method to identify the flow. FleXam [12] added a sampling mechanism in OpenFlow that allowed the controller to obtain traffic statistics.

III. SDN Measurement Architecture on Sampling

In the following, we discuss the design of *eOpenFlow*.

The sampling-capable SDN Measurement System is shown in Figure 1. An application plane receives and parses user defined measurement tasks. Based on the analysis of the requested measurement, the application plane determines the targeted sample objects and delegates measurement tasks to selected switches, invokes the traffic sampling module of control plane to delivery the requested measurement results.

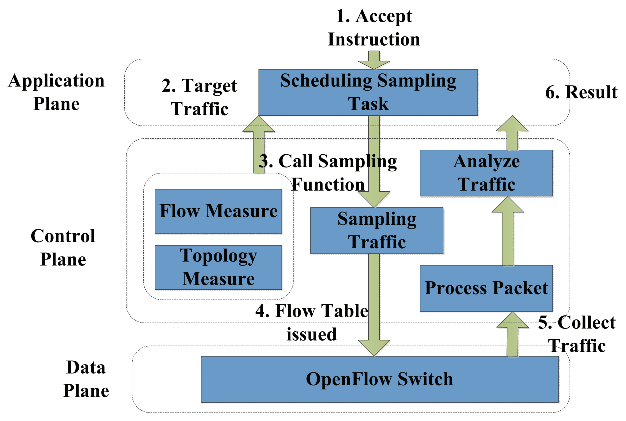

Fig. 1. The Sampling-capable SDN Measurement System

The traffic sampling module of control plane generates and sends the corresponding flow table rules with the sampling action OFPAT_OUTPUT_SAMPLING according to the requirements of the application plane sampling task scheduling. After sending out the rules of sampling flow table, the sample processing module is responsible for collecting and pre-processing the traffic, collecting the sampling results from delegated switches and delivering the traffic samples to the flow measurement module according to the measurement requirements of the sampling tasks. The flow measurement module extracts the flow statistics information from the inputting sample packets and calculates the corresponding results.

Because the traffic sampling function on SDN switch is matched by flow tables, in order to ensure normal flow forwarding on the switch and avoid the interference of the normal flow table matching, we design a new extension OpenFlow protocol in both multi-level flow table and group table for different measurement needs. We also define the OpenFlow sampling action in sampling flow tables.

A. *Packets are sampled against Multiple Tables*

OpenFlow 1.1 and subsequent versions propose the design of multistage flow tables in OpenFlow switch. The multi-level flow table can decompose the complex flow table items, and adopt the pipelined method. Each level of the flow table can perform different actions, and increase the flexibility of flow table. As shown in Figure 2, when a packet arrives at a switch, it first matches the first-level flow table and performs normal forwarding based on the action set of the flow table entry. The data packet then jumps to the flow table entry containing the sampling action according to the *Goto* instruction in the first-level flow table, and performs the sampling action to complete the traffic sampling.

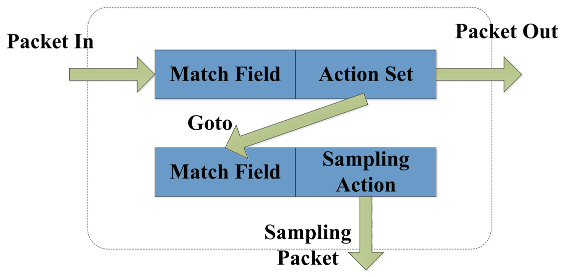

Fig. 2. Packets are Sampled against Multiple Tables

B. *Packets are Sampled against Group Tables*

OpenFlow 1.1 and subsequent versions also present the concept of group tables that contain group identifier, group type, counters, and action buckets. Each action bucket can define different execution actions and parameters. A group table in OpenFlow has two purposes: on the one hand, group table can make different flow table items to implement the same set of actions in order to improve switch flow matching and processing efficiency; on the other hand, the group table in different action bucket which can further classify the flow and enrich the functions of the flow table rules. When the controller needs to sample the traffic of multiple flows or designated ports, the sampling rule based on the group table can be adopted. In Figure 3, the sampling action is sent to the action list of the group table by the controller. When a packet arrives at the switch, it first matches the corresponding flow table entry according to the rule, and then performs regular forwarding and other operations. If the packet needs to be sampled, it will jump to the corresponding group table according to the group table identifier to perform the preset sampling operation.

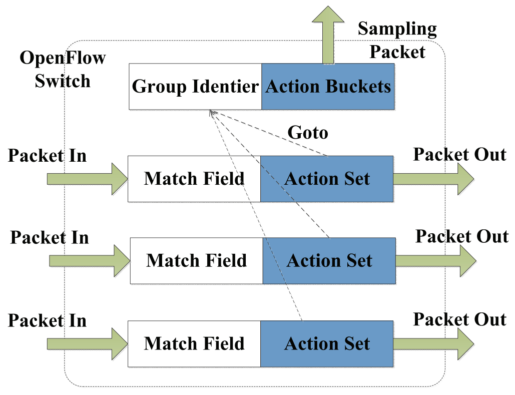

Fig. 3. Packets are Sampled against Group Tables

```
struct ofp_action_output_sampling {
  ovs_be16    type;
  ovs_be16    len;
  ovs_be32    port; /*Packet Forwarding Port*/
  ovs_be16    max_len;
  uint8_t  p;     /*Random sampling parameters p*/
  uint8_t  m;     /*System sampling parameters m*/
  uint8_t  n;     /*System sampling parameters n*/
  uint8_t  pad[5];
}
```

Fig. 4. OFPAT_OUTPUT_SAMPLING Data Structure

Compared with the sampling rules based on a multistage flow table, the sampling rules based on a group table have better flexibility. When multiple flows apply the same sampling method and sampling rate, the group table can reduce the number of sampling flow table entries, improve the efficiency of a switch. At the same time, the group table does not directly match the data packets, and thus the controller can preset multiple sampling methods and sampling rate. Group table method can be deployed with the sampling rules on a switch for a longer time to achieve long-term sampling and monitoring.

C. Defined OpenFlow Sampling Action

In the structure of OpenFlow protocol, each field may have different values on behalf of the function of the structure. Therefore, we can expand the value of the parameter field to achieve the definition of new features. At the same time, the OpenFlow structure contains both valid fields and free fields for padding bytes. Such OpenFlow structure accommodates enough room to extend OpenFlow functions by adding valid fields. Since traffic sampling action needs to process data packets directly, the most suitable modification method is to extend the flow table functions in OpenFlow switches.

First of all, we reconstruct the action set of flow table. OpenFlow 1.3 defines several action types for the action set of flow table entries. Each action has a default value of 27. We have extended the *value* field based on the existing action type, and added the sampling action OFPAT_OUTPUT_SAMPLING. In the next, we extend the defined OFPAT_OUTPUT_SAMPLING action with its data structure shown in Figure 4.

OFPAT_OUTPUT_SAMPLING refers to the data structure of the OFPAT_OUTPUT action. The forwarding port is defined. The sampling result can be forwarded from any port of a switch or directly to the controller for collection. OFPAT_OUTPUT_SAMPLING pad array for the *fill* field. The use of the field storage space can increase a number of parameters, such as the selection of sampling methods, sampling rate function. OFPAT_OUTPUT_SAMPLING defines two sampling methods: 1) Random Sampling: sampling the probability of the target through the switch traffic, where $0 \leq p \leq 1$; (2) System Sampling: the target traffic through the switch every m packets before the extraction of n packets.

The method of implementing sampling function by extending the action set of flow table has three advantages: (1) Traffic sampling is decentralized to the switch implementation, the upper layer only needs to collect the sampling results, which reduces the extra message sampling and transmission resulting in communication load; (2) The new sampling action does not affect the regular functions of OpenFlow protocol, but also has good compatibility in several versions of OpenFlow protocol; (3) The method of extending action set is easier to implement in switch, and does not affect the original structure of switch, and thus becomes easier to implement more traffic sampling algorithms.

IV. Implementation of Sampling Functionality

A. Extension to OpenvSwitch

We choose OpenVSwitch 2.3.1 [14] as the code base for the development. The flow sampling function is mainly implemented in the *vswitchd* module as shown in Figure 5. When a packet matches the sample flow table in the datapath, it enters the user space. The *vswitchd* module invokes the sampling function according to the parameters of the sampling behavior OFPAT_OUTPUT_SAMPLING and samples it according to a preset sampling rate. Finally, the sampled packets are sent to the controller or forwarded to other traffic collection devices according to the forwarding port preset in the OFPAT_OUTPUT_SAMPLING action. When performing sampling, the sampled target traffic has already been filtered through the flow table entry at the system kernel. The sampling process is only for the specified target stream, which does not process incoming traffic unrelated to the target payload.

We have both random sampling and system sampling implemented respectively. Random sampling generates a random number between 0 and 1 for each packet matching the sampling flow table. If the random number is smaller than the preset sampling rate p, the packet is extracted. Random numbers are obtained by varying the random seed, and can be extended to generate random numbers of different distributions as needed. In order to reduce memory consumption, we use

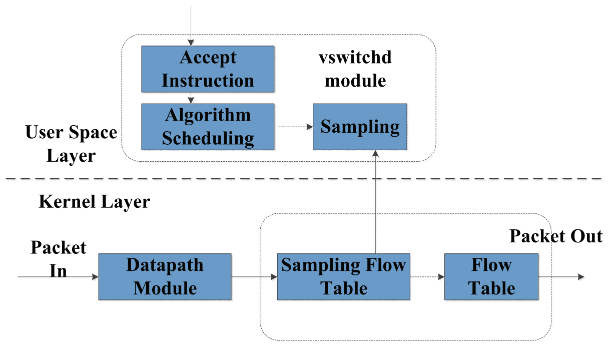

Fig. 5. System Implementation

the rx_packets statistic of the counters of flow table. When packets entering user space of a OpenVSwitch, and if the current $rx\_packets\%m < n$, the packet is extracted.

B. Traffic Sampling on Controller

TrafficSampling module is defined to provide traffic sampling functionality. This module needs to listen for the FLOW_REMOVED message returned from a switch when it is loaded. The Traffic Sampling module contains the TrafficSampling class and the TrafficSamplingREST class. The TrafficSamplingREST class encapsulates the sample instruction as a REST interface for the application layer's method calls. The TrafficSampling class implements the function of sampling the specified traffic. The main functions are shown in Table I. It is used to send sampling instructions and maintain the sampling tasks.

Function Name	Description
addSamplingTask (match, type, rate)	generate traffic sampling tasks
getTargetSwitch (match)	Select the sampling switch
send_sampling_flow (datapath, match, type, rate)	issue the sample flow table
samplingMonitor()	monitor and maintain the sampling tasks
flow_removed_handler (ev)	listen to the FLOW_REMOVED message and end the sampling task

TABLE I
The Main Method of Flow Sampling Module

The implementation of traffic sampling includes initializing parameters, assigning sampling tasks, and maintaining sampling tasks. The *addSamplingTask* method is used to obtain sample requirements, including the attributes of the target traffic, the sampling method, and the sampling rate. The *getTargetSwitch* method obtains the forwarding path of the target traffic from the topology measurement module and selects the switch that performs sampling method.

Issued sample tasks needs to use send_sampling_flow method, which generates the FLOW_MOD message with the sampling action OFPAT_OUTPUT_SAMPLING according to the related parameters and sets the timeout of the flow table entry. Then the send_sampling_flow method sends the FLOW_MOD message to the target switch, and adds the sampling task to the task schedule. During the executing a sampling task, the flow_removed_handler method listens for a FLOW_REMOVED message from the switch. If the flow table with the sampling action expires, then the sampling task ends.

C. Processing Sampling Packets

We define the *SamplingCollector* module, which contains the *SamplingCollector* class. The main functions are shown in Table II. When the sample processing module runs, it generates a main thread that is responsible for receiving information about the sampling task from the traffic sampling module. Sample processing is done by three sub-threads: (1) monitoring and collecting the sampling results, (2) sorting the traffic samples according to the sampling task, and (3) finally outputting or storing the traffic samples.

Function Name	Description
addCollectingTask (dpid, flows)	generate sampling task description
flowCollector()	listen and collect the sampled results from the switch
flowAnalyzer()	samples are analyzed by sampling task
flowStorage()	output and store traffic samples

TABLE II
The Main Method of Sampling Collector Module

After running the sample processing module, three sub-threads for traffic collection, sample classification, and sample storage are created. These sub-threads are executed concurrently at run-time to collect the sampling results. When the traffic sampling module generates a sampling task, it reports the attributes of the sampling task to the sample processing module, such as the datapath id of the sampling switch, the target traffic attribute. The traffic collection sub-thread will continue to receive sampling results from the target switch and collect the time for each packet until the sampling task on the switch ends.

At the same time, the sample classification sub-thread will classify all traffic samples. Since the sampling task is performed at the granularity of the switch, there may be multiple measurement tasks on a switch. The same measurement task may sample multiple flows. Therefore, the sample classification sub-thread needs to maintain the traffic sample flow on a switch Information table, and analyze the flow information according to the sampling task.

The flow storage sub-thread periodically stores the sample traffic by examining the flow information tables maintained by the sample classification sub-threads and storing the categorized traffic samples for sample measurement and other functions at the application layer. When the three sub-thread tasks are completed, the sample processing module's main thread will initialize the three sub-thread, and release the used memory.

D. Sampling Measurement

The *SamplingMeasure* module is implemented for sampling measurement. The module contains the *SamplingMeasure* class and *SamplingMeasureREST* class, which is responsible for the sampling measurement function package for the REST interface to call the application layer method. The sampling measurement module comprises several sub-modules which are respectively responsible for measuring the functions of the task scheduling, the sample estimation, the throughput measurement and the packet loss rate measurement. The throughput measurement sub-module comprises the throughput measurement for the port and the flow for the specific switch. The packet rate measurement sub-module includes flow measuring.

After the sample measurement module is running, it first establishes four sub-threads for (1) task scheduling, (2) sample estimation, (3) throughput measurement, and (4) packet loss rate measurement. After the task scheduling thread receives the measurement task request from the upper layer, it collects the traffic by calling the traffic sampling module and the sample processing module. At the same time, the sample estimation thread reads the sampling result output by the sample processing module, the statistics such as the number of packets and bytes of the sample information, estimate the number of bytes and the number of packets in the original period. The results are stored in the corresponding memory space and output to the corresponding measuring thread.

V. Experimental Results

A. Testing the Sampling Fuction

In this section, we present the evaluation results via various experiments. The experimental topology is shown in Figure 6. In the experiment, we use iperf tool to generate the four groups of UDP traffic as shown in Table III, and perform a random sampling of 15% for stream No.1 through H1 switch (from H1 to H4). The sampling task is executed 30 seconds, and the sampling switch is changed around the 15th second. The flow sampling results are shown in Figure 7.

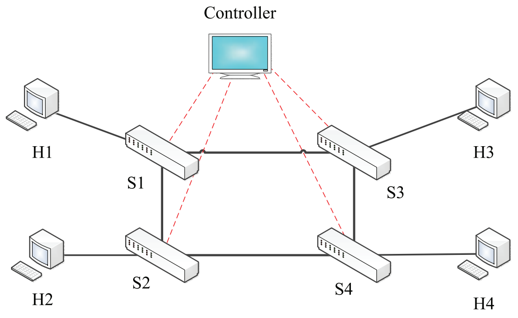

Fig. 6. Topology of Experimental Network

Figure 7 shows the collected traffic per second from H1 to H4. The first 15 seconds of sampling traffic comes from switch S3, and the last 15 seconds of sampling traffic comes from switch S4. We can see from Figure 7 that the entire measurement task is not affected by the change sampling switch, and the collected sample per second is fluctuating at the theoretical value (1.5 Mbits). This result shows that the traffic sampling module can run normally and perform traffic sampling, and has the ability to maintain the normal operation of the sampling task by changing the sampling switch.

No.	Source	Destination	Path	Speed
1	H1	H4	S1-S3-S4	10Mbits
2	H4	H1	S4-S3-S1	5Mbits
3	H3	H1	S3-S1	3Mbits
4	H2	H4	S2-S4	8Mbits

TABLE III
Sampling Parameters

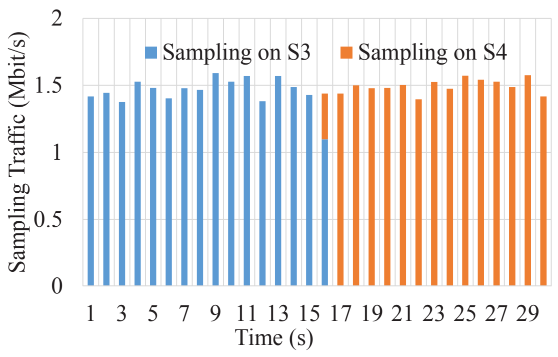

Fig. 7. Experimental Results

We use real network traffic to verify the sampling functionality. We use the Tcpreplay tool to generate traffic and play back the captured traffic. We replay the captured pcap file in the Mininet experiment network to generate the real network traffic, and use both the random sampling and the system sampling methods to sample traffic with 1/20 sampling ratio.

Sample	Mean	Standard Deviation
Total Traffic	448.9781	575.7459
Random Sampling	449.2905	576.2886
System Sampling	448.9781	575.7459

TABLE IV
Experimental Result of Packet Average Length

We compare the packet length distributions of the two groups samples and the total traffic, and the analyzed results are shown in Table IV. Table IV shows that the packet average length and the standard deviation of the two groups are matching the distribution of original traffic.

B. Measuring Throughput

In this experiment, we use the same network topology as Figure 6. We test the throughput from a given switch port and the throughput of a given flow. We use the H1-S1-S2-H2 path for port throughput experiments where port 2 of S1 connects to port 1 of S2.

The experiment uses iperf to generate UDP packets with an initial rate of 5 Mbits from H1 to H2, and then increases the traffic rate to 25 Mbits every 10 Mbits. The duration of each traffic is 10 seconds. The sample measurement module sets up the throughput measurement task, sampling the flow of the port 1 in the S2 switch with 1/10 sampling ratio , and calculating and outputting the throughput change of the port. We test the throughput measurement module and compare the sample measurements. Figure 8 shows the traffic throughput results over time at the given port. Figure 8 shows that the relative error in the results of the sampling measurement, which is related to the errors caused by the collected traffic as well as the total traffic. The relative error between the measured results and iperf preset flow rate is within 15% when the generated throughput is from 5 Mbits to 25 Mbits, and the average accuracy is over 90%, so the relative error is not affected by the traffic size. We can adjust the sampling ratio to control the application error in the commercial network.

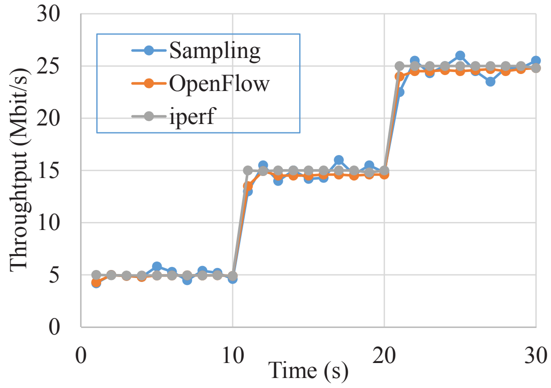

Fig. 8. Experimental Result of Throughput on Given Port

The throughput measurement of a given flow requires multiple sets of traffic to be generated in the experiment network, and the throughput changes of different flows are measured separately. The sample measurement module sends a group table with sampling action at switch S3 with 110 sampling ratio. iperf tool is used to generate three UDP traffic with 5s time interval. The first flows is from H1 to H4 at a rate of 5 Mbits. The second flows is from H3 to H1 at a rate of 15 Mbits, and The third flows is from H2 to H4 at a rate of 10 Mbits. The three curves in Figure 9 show throughput over time. The sample measurement module can measure throughput individually for each flow, and can measure the real-time status of network traffic.

VI. Conclusion

Based on the function of OpenFlow protocol and its expansibility, we design and implement a SDN network measurement system with traffic sampling function. A new sampling action, OFPAT_OUTPUT_SAMPLING, is added to the controller to generate the sampling rules based on either multi-level flow table or group table. A specified flow is sampled on the OpenFlow switch; and a control collects and analyzes the samples to realize the measurement function. The system components are realized by the RYU controller and OpenvSwitch. Finally, a series of experiments are carried out to test the SDN network measurement system, and all demonstrate positive results of the proposed approach (i.e., eOpenFlow).

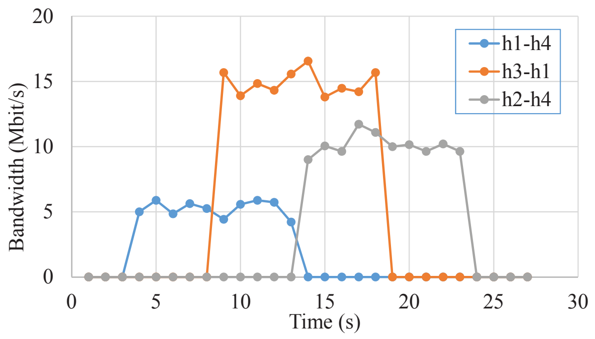

Fig. 9. Experimental Result of Flow Throughput

Acknowledgment

This work was supported by the National High Technology Research and Development Program (863 Program) of China (2015AA010201); and six talent peaks of the high level Talents Project of Jiangsu province (2011-DZ024).

References

[1] Yu C, Lumezanu C, Zhang Y, et al. Flowsense: Monitoring network utilization with zero measurement cost[C]//Passive and Active Measurement. Springer Berlin Heidelberg, 2013: 31-41.

[2] Tootoonchian A, Ghobadi M, Ganjali Y. OpenTM: traffic matrix estimator for OpenFlow networks[C]//Passive and active measurement. Springer Berlin Heidelberg, 2010: 201-210.

[3] Chowdhury S R, Bari M F, Ahmed R, et al. Payless: A low cost network monitoring framework for software defined networks[C]//Network Operations and Management Symposium (NOMS), 2014 IEEE. IEEE, 2014: 1-9.

[4] Su Z, Wang T, Xia Y, et al. CeMon: A cost-effective flow monitoring system in software defined networks[J]. Computer Networks, 2015, 92: 101-115.

[5] Van Adrichem N L M, Doerr C, Kuipers F A. Opennetmon: Network monitoring in openflow software-defined networks[C]//Network Operations and Management Symposium (NOMS), 2014 IEEE. IEEE, 2014: 1-8.

[6] Sahri N M, Okamura K. Adaptive Anomaly Detection for SDN[J]. Proceedings of the Asia-Pacific Advanced Network, 2015, 40: 55-59.

[7] Moshref M, Yu M, Govindan R. Resource/accuracy tradeoffs in software-defined measurement[C]//Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013: 73-78.

[8] Curtis A R, Mogul J C, Tourrilhes J, et al. DevoFlow: scaling flow management for high-performance networks[C]//ACM SIGCOMM Computer Communication Review. ACM, 2011, 41(4): 254-265.

[9] Yu M, Jose L, Miao R. Software Dened Trafc Measurement with OpenSketch[C]//Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). 2013: 29-42.

[10] Suh J, Kwon T T, Dixon C, et al. OpenSample: A low-latency, sampling-based measurement platform for commodity SDN[C]//Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on. IEEE, 2014: 228-237.

[11] Afek Y, Bremler-Barr A, Landau Feibish S, et al. Sampling and Large Flow Detection in SDN[C]//Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. ACM, 2015: 345-346.

[12] Shirali-Shahreza S, Ganjali Y. Traffic statistics collection with FleXam[C]//ACM SIGCOMM Computer Communication Review. ACM, 2014, 44(4): 117-118.

[13] Yassine A, Rahimi H, Shirmohammadi S. Software defined network traffic measurement: Current trends and challenges[J]. Instrumentation & Measurement Magazine, IEEE, 2015, 18(2): 42-50.

[14] Open vSwtich[CP/OL]. http://www.openvswitch.org.