Getting traffic statistics from network devices in an SDN environment using OpenFlow

Diyar Jamal Hamad , Khirota Gorgees Yalda, and Ibrahim Tanner Okumus

Abstract—Software Defined Networking (SDN) provides a significant advantage because it is easier to tune up and introduce new functionalities. Although there are previous work focusing on the problem of network management, monitoring and control to improve the QoS seen by the customers, only some supply solutions for measuring network performance, e.g., latency, throughput, and packet loss. In this study we introduce how to get traffic measurement statistics from network devices in an SDN environment. In terms of network monitoring, OpenFlow allows to build a monitoring solution adjusted to the specific network needs. We demonstrate how to use OpenFlow features to get traffic statistics from network devices. In our test environment, collected traffic statistics are used to calculate the available bandwidth on each link in the network. We analyse the effect of statistics collection frequency on the network load and the accuracy of the results collected.

Keywords— SDN/OpenFlow1.3, Floodlight controller, Mininet, CPqD Switch, and Network virtualization.

I. INTRODUCTION

Generally, it is difficult to experiment with new ideas in actual networks because new ideas often include nonstandard aspects. It is difficult to incorporate these changes into an existing network since the devices in traditional networks do not allow changes to be made in their software systems.

Software Defined Networking (SDN) separates the control plane from the data plane. Control plane functionalities are handled by an entity called controller which can be centralized or distributed. Controller communicates with network devices to collect information from them and also to push configuration information to them. One controller can manage many network devices in the network.

Controller receives the information from all switches in the network and based on the received information, a controller can build the network topology. Then switches receive flow table from the controller to operate properly based on the flow table. If a switch receives a packet from a flow which does not have a matching entry in the flow table, then the switch transmits the packet to the controller. After receiving it, based on the packet information, the controller generates a forwarding rule and then sends the forwarding rule to the switch. A controller can add, update, and delete flow entries in flow tables, both reactively and proactively.

In SDN terminology communication that occurs between controller and network devices is called southbound communication. One mechanism that allows the control plane to communicate with networking devices is OpenFlow. OpenFlow is an open source framework created in 2008 at Stanford University and from 2011 developed within the Open Networking Foundation (ONF). Its aim is to "open" the control of network nodes to allow the separation of forwarding plane and control plane.

OpenFlow standard has different versions. In this study we focused on OpenFlow 1.3 to investigate the use of metering and measurement functionalities. OpenFlow 1.3 introduces new features for monitoring and operations and management (OAM). To that end, the meter table is added to the switch architecture. Figure 1 shows the structure of meter table entries. A meter is directly attached to a flow table entry by its meter identifier and measures the rate of packets assigned to it. A meter band may be used to rate-limit the associated packet or data rate by dropping packets when a specified rate is exceeded. Instead of dropping packets, a meter band may optionally recolour such packets by modifying their differentiated services (DS) field. Thus, simple or complex QoS frameworks can be implemented with OpenFlow 1.3 and later specifications.

Meter Identifier	Meter Bands	Counters

Figure 1: Meter table entry.

In this study, we examine the traffic statistics collection features of OpenFlow protocol. Using the provided features we calculate the actual link utilizations and available link capacities in different time granularities in a test environment. Using the results, we analyse the effect of statistics collection frequency on the network load and the accuracy of the results. Collecting bandwidth measurement statistics on switches offers bandwidth assurance for designated traffic [1]. Based on the traffic unit, bandwidth provisioning can be at different granularity levels [2]. Port-level bandwidth provisioning assures bandwidth for the traffic from an input port to an output port.

II. RELATED WORK

The OpenFlow protocol provides functions to query switches for the number of packets or bytes in flows matching a specific rule or passing a specific port. Prior work relies on this capability to compute utilization in the network [3]. OpenTM measures networks-wide traffic matrix by periodically polling one switch on each flow's path and then combining the measurements. Polling a single switch does not impose material load on the network but may affect accuracy if the switch is not carefully chosen.

Jose et al [4] detect heavy hitters by continually adapting polling rules to focus on the flows that are more likely to have high volume. Both approaches have to carefully schedule measurements to limit the polling overhead. The authors in [5] showed that centralized controller architecture will interrupt network traffic and flow requests in case of controller failure. Specifically, they proposed a distributed architecture SiBF, which consists of an army of rack managers (RMs), one per rack, acting as controllers. Consequently, when the master controller fails, flow requests are handled by another standby controller (RM) until the master controller comes back up. In case of switch failure, SiBF installs new mappings (new back-up flow entries) in the ToR switches for each active entry.

sFlow [6] works similarly and has the advantage to let the agents "push" their counters. This means that fewer packets are needed to obtain the relevant data as there is no request. An sFlow traffic analyser is still needed to collect the data and an agent must be put in every switch.

Other initiatives such as OpenSAFE uses traffic duplication to monitor the network adding a very high overhead while FlowSense[7] uses a push mechanism to analyse link utilization passively.

The authors in [9] implement OpenNetMon to monitor latency, throughput and packet loss in OpenFlow networks. This application allows determining online whether the end-to-end QoS parameters are satisfactory. Then, the application sends the data relative to throughput, delay and packet loss to the controllers for Traffic Engineering (TE) purposes. The throughput and packet loss are obtained from polling flow source and destination switches.

III. MONITORING IN SDN

Traditionally, many different monitoring techniques are used in computer networks. Every measurement technique requires a separate hardware installation or software configuration, making it a tedious and expensive task to implement. However, OpenFlow provides necessary interfaces to implement most of the discussed methods without the need of customization.

By using some of the OpenFlow protocol messages described in the OpenFlow specification [8], statistics collection can be achieved and calculations can be made on the collected data for specific purposes such as delay estimation etc. The two types of messages that are provided by OpenFlow are:

- STATISTICS REQUEST: Message sent from the controller to a switch requesting its current set of statistics for flows, ports, etc.
- STATISTICS REPLY: Message sent from a switch to the controller, in reply to a request message.

Network measurement methods are generally divided into two classes: passive and active methods. Passive measurement methods measure network traffic by monitoring traffic for specific message types to infer measurement values. The advantage of passive measurements is that they do not generate additional network overhead, and thus do not influence network performance. Unfortunately, passive measurements rely on installing in-network traffic monitors, which is not feasible for all networks and require large investments. Active measurements on the other hand inject additional packets into the network, monitoring their behaviour. For example, the popular application ping uses ICMP packets to reliably determine end-to-end connection status and compute a paths round-trip time [10].

Both active and passive measurement schemes are useful to monitor network traffic and to collect statistics. However, one needs to carefully decide which type of measurement to use. For instance active measurements introduce additional network load affecting the network and therefore influence the accuracy of the measurements themselves.

Often network measurements are performed on different layers. Measurements on the application layer are preferred to accurately measure application performance. Network layer measurements use infrastructure components (such as routers and switches) to obtain statistics. This approach is not considered sufficient for some as the measurement granularity is often limited to port based counters. It lacks the ability to differ between different applications and traffic flows.

The Simple Network Management Protocol (SNMP) is one of the most used protocols to monitor network status. Among others, SNMP can be used to request per-interface port-counters and overall node statistics from a switch. Being developed in 1988, it is implemented in most network devices.

In this study our goal to get actual network usage information that can be used in different parts of the network management layer to provision the network online. Thus our aim is to get raw link usage information from the network devices independent of the application layer details. For each link on the network, we collect actual link usage in terms of number of bytes. We use this information to calculate near real-time available bandwidth on each link. This information will help us with deploying custom routing protocols, provision the network, load-balancing, and quality of service purposes.

IV. TEST ENVIRONMENT AND EVALUATION RESULTS

In our test environment we used different components to create an SDN network. Figure 2 shows the sample topology used in our tests.

We used mininet to create SDN capable test topology. Mininet [11] is an open source network simulator for modelling software defined networks. One of OpenFlow soft switch is cPqD [12]. The CPqD switch is a version of the Stanford Reference Implementation which has been updated to support OpenFlow 1.3. CPqD supports groups and meters. CPqD switch was written in Linux. It's implemented in operating system's user space. This approach gives a lot of flexibility and allows quick development and testing of new OpenFlow features. OpenFlow Softswitch supports OpenFlow 1.3 and multiple controller & redundancy.

The OpenFlow network controller used in the experiment is Floodlight. Many SDN controllers have been developed since the introduction of SDN [13]. However, one of the most widespread controllers is Floodlight. Floodlight is Java-based open source software based on the Beacon controller implementation developed at the Stanford University that works with physical and virtual OpenFlow switches. The last release of Floodlight is the version 1.1.

In order to evaluate the suggested methods, one or multiple UDP flows are created between H1 towards H3 with data rate of 5 Mbps, through the testbed. The experimental evaluation aims to prove the system's accuracy and analyse the overhead of the method used.

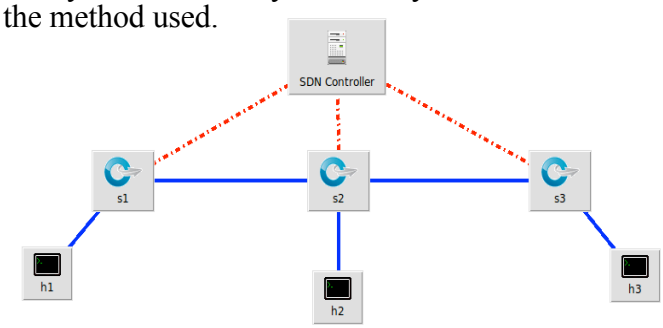

Figure 2: Evaluation Topology

In the OpenFlow switch specifications [14] it is stated that switches have to keep counters for port, flow table/entry, queue, group, group bucket, meter and meter band. Table 1 presents the Per Flow Entry counters used in this paper. Furthermore, in order to follow the statistics for more than one flow, there is an option to bundle multiple flows in a group and observe their aggregated statistics.

Counter	Description
Received Packets	Counts the number of packets
Received Bytes	Counts the number of bytes
Duration(seconds)	Indicates the time the flow has been installed on the switch in seconds
Duration(nanoseconds)	Counts time the flow has been alive beyond the seconds in the above counter

Table 1. Counters [14].

We retrieve the available bandwidth related to a specific link by means of a number of packets passed through the switch port connected to it.

Specifically, a number of bytes transmitted or received from/to a port are related to the low-level data transmission (i.e., throughput, not the goodput). Our architecture allows us to continuously sample, for each switch port, the transmitted or received bytes with a specific frequency. Hence, comparing the retrieved values in two different instants, it is possible to approximately know the bandwidth usage of the link connected to that port.

Both the OpenFlow switch and the OpenFlow controller must accept any OpenFlow message types and sub-types on all connections. The main connection or an auxiliary connection cannot be restricted to a specific message type or sub-type. However, the processing performance of different message types or sub-types on different connections may be different.

OpenFlow 1.3 supports meters. It is possible to set multiple meter bands per meter and submit it to switch. In Floodlight controller, this can be achieved by using IOFSwitchListener specifically in switchAdded class that's included in IOFSwitchListener interface. When the MeterMod including MeterBand is set, ofsoftswitch(CPqD) needs to be used because currently only this softswitch is supporting Meters in OpenFlow v1.3. After setting the meters to switches, flows need to be assigned to meters so that meters can identify which packets are intended for that meter. In order to get switch statistics, controller sends an OFMeterStatsRequest to relevant switch with meter identification included and the switch will reply with OFMeterStatsReply its statistics. One of the items received in the statistics will be the total received bytes matched by each flow. Let's assume at time t1 statistics information is received and the total received byte count is Bt1. If this process is repeated periodically it is possible to calculate the utilized bandwidth for that specific meter. After a period (P) another OFMeterStatsRequest is sent and the recived byte count is Bt2. This time received byte count should be larger than the previous number (ignoring count overflows). Utilized bandwidth (UB) in bits per second can be calculated as:

$$UB = [(B_{t2} - B_{t1}) / P] * [8 \text{ bits}]$$

UB is the bandwidth consumed by this flow on the corresponding meter. On the controller a thread can be set to repeat the process on defined periods to give us an average bandwidth consumption of that flow over the time period we chose.

OpenFlow ports are the network interfaces for transit packets between OpenFlow processing and the rest of the network. OpenFlow switches connect logically to each other by their OpenFlow ports. An OpenFlow switch has a number of OpenFlow ports available for OpenFlow processing. The set of OpenFlow ports may not be identical to the set of network interfaces submitted by the switch hardware. Some network interfaces may be not enabled for OpenFlow, and the OpenFlow switch may define extra OpenFlow ports.

As ports are added, modified, and removed from the datapath, the controller needs to be informed with the OFPT_PORT_STATS request message.

The switch and controller can verify proper connectivity through OpenFlow protocol with echo request (OFPT_ECHO_REQUEST) and reply (OFPT_ECHO_REPLY) messages [15]. The body of the message consist of uninterpreted data that is to be echoed back to the requester. The requester matches the reply with the transaction id from the OpenFlow header.

In order to get switch statistics OpenFlow provides port level functionalities. Using the defined methods, it is possible to get detailed statistics of a specific port on a selected switch. When a switch receives an "OFPortStatsRequest" message it generates a response, which is the "OFPortStatsReply" message. This message contains the information obtained from the switch counters [14]. On flow level it gives the duration of the flow (in nanoseconds), packet and byte count. Port statistics give more information about the state (both transmitted and received) such as number of dropped packets, bytes, errors and collisions. The controller obtains information for every flow that follows the same path. Per-flow port byte is calculated by subtracting the increase of the OpenFlow port bytes counter of the source switch with that of the destination switch [16].

Port statistics provide raw utilization data on each link. OFPortStatsReply messages contain received byte count along with other statistics data. Received byte information can be used to calculate link utilization. Formula 1 provided above for meters can be used again for the same purpose. Periodically polling switches will provide necessary information to calculate link utilization. Granularity of the polling period can be changed to get more timely results in time-critical network management functionalities. This information can be used for traffic engineering practices.

On the test topology shown in Figure 2, traffic is generated between two hosts host1 (H1) and host3 (H3). To test the accuracy and the performance different flow durations are used. Generated flows

are chosen as UDP flows to ensure the stable bit rate on the link. Each UDP flow is adjusted to have 5Mbps of traffic. First flow was active for 60 seconds second flow 300 seconds and third flow 600 seconds. As the flow duration increased, number of polls also increased and that information is used to analyse the extra load injected into the network for statistics collection.

Information about ports statistics is requested with the OFPMP_PORT_STATS multipart request type (sizeof(struct ofp_port_stats_request) == 8.) The port_no field optionally filters the stats request to the given port. To request all port statistics, port_no must be set to OFPP_ANY (sizeof(struct ofp_port_stats) == 112).

The duration_sec and duration_nsec fields indicate the elapsed time the port has been configured into the OpenFlow pipeline. The total duration in nanoseconds can be computed as durationsec_10^9 + duration nsec. Implementations are required to provide second precision; higher precision is encouraged where available.

The port description request OFPMP_PORT_DESCRIPTION enables the controller to get a description of all the ports in the system that supports OpenFlow. The request body is empty. The reply body consists of an array of the following:

```
/* Description of a port */
struct ofp_port {
uint32_t port_no;
uint8_t pad[4];
uint8_t hw_addr[OFP_ETH_ALEN];
uint8_t pad2[2]; /* Align to 64 bits. */
char name[OFP_MAX_PORT_NAME_LEN]; /* Null-terminated */
uint32_t config; /* Bitmap of OFPPC_* flags. */
uint32_t state; /* Bitmap of OFPPS_* flags. */
/* Bitmaps of OFPPF_* that describe features. All bits zeroed if
* unsupported or unavailable. */
uint32_t curr; /* Current features. */
uint32_t advertised; /* Features being advertised by the port. */
uint32_t supported; /* Features supported by the port. */
uint32_t peer; /* Features advertised by peer. */
uint32_t curr_speed; /* Current port bitrate in kbps. */
uint32_t max_speed; /* Max port bitrate in kbps */
};
sizeof(struct ofp_port) == 64.
```

When we generated traffic between H1 and H3 in our topology by using Iperf commands [17], the message size of Port Statistics Request in wireshark is 86 Kb and the Reply is 390 Kb.

Figure 3, represent utilized bandwidth for each interswitch port between Switches with 5Mb traffic and 60 sec.

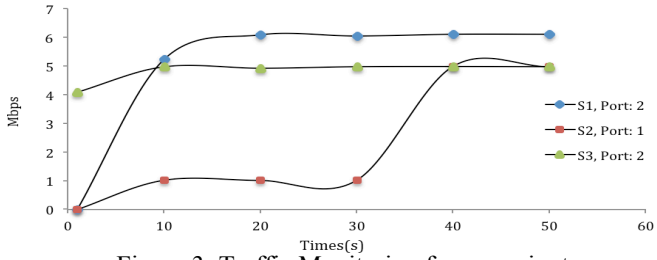

Figure 3: Traffic Monitoring for one minute

Figure 4, represent utilized bandwidth for each port between Switches with 5Mb traffic and 300 sec.

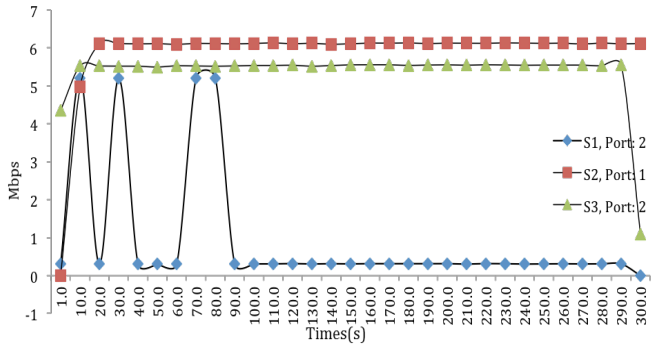

Figure 4: Traffic Monitoring for 5 minutes

Figure 5, represent Utilized bandwidth for each port between Switches with 5Mb traffic and 600 sec.

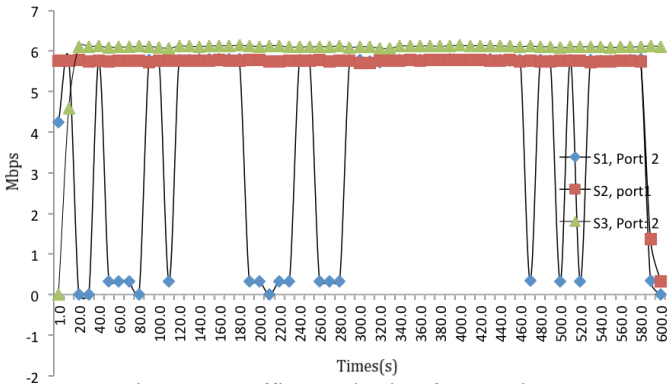

Figure 5: Traffic Monitoring for 10 minutes

Figures show that the port statistics collected from switches using OpenFlow messages follow the generated traffic rate closely. The rate received from ports is a little bit higher than the generated traffic. The reason for that is apart from generated traffic, there is background traffic such as control traffic in the network.

V. CONCLUSION

Most published OpenFlow/SDN use cases highlight the multi-faceted areas of applications that exist for these kinds of networks. We presented a way to efficiently infer bytes_in_count in an SDN environment by capturing and analysing OpenFlow Stats request and Reply messages between switches and controller for both of Port and Meter features. On a virtual OpenFlow testbed we displayed that our method is accurate and provides updated link utilization information through statistics Request and Reply messages. According to collected information in Floodlight controller, this design can be used to flollow throughput in interswitch links which in turn can be used to calculate other traffic engineering related parameters such as available link capacity on the links. This approach presented in this study shows SDN/OpenFlow architecture presents features that can be readily used to implement fine grained traffic engineering policies in actual networks that support SDN/OpenFlow.

REFERENCES

[1] S. Chuang, A. Goel, N. McKeown, and B. Prabhkar, “Matching output queueing with a combined input output queued switch,” IEEE INFOCOM, New York, Mar. 1999.

[2] S. Chuang, S. Iyer, and N. McKeown, “Practical algorithms for performance guarantees in buffered crossbars,” IEEE INFOCOM,Miami,FL, Mar. 2005.

[3] OpenDaylight. Available online: http://www.opendaylight.org/ (accessed on 22 February 2014).

[4] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, OpenTM: Traffic Matrix Estimator for OpenFlow Networks. In PAM, 2010.

[5] L. Jose,M. Yu, and J. Rexford. Online measurement of large traffic aggregates on commodity switches. In USENIX Hot-ICE, 2011.

[6] Macapuna, C.A.B., Rothenberg, C.E., Magalhaes, M.F., "In-Packet Bloom Filter-Based Data-Center Networking with Distributed OpenFlow Controllers, "IEEE 2010GLOBECOM Workshops, pp.584– 588, 6–10 December 2010.

[7] Yu, Curtis, et al., “FlowSense: Monitoring Network Utilization with Zero Measurement Cost.” Passive and Active Measurement, Springer Berlin Heidelberg, 2013.

[8] OpenFlow Switch Specification, Version 1.3.0. 2013. Available online:https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf (accessed on 27 Sept 2015)

[9] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, “OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks,” in NOMS , pp. 1–8, 2014.

[10] Erickson, D. The Beacon OpenFlow Controller. In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Hong Kong, China, 12–16 August 2013; pp. 13–18.

[11] Mininet. Available: http://mininet.org/.

[12] CPqD switch is available online: https://github.com/TrafficLab/of13softswitch.

[13] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, “A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks,” Communications Surveys Tutorials, IEEE , vol. PP, no. 99, pp. 1–18, 2014.

[14] OpenFlow Switch Specification, Version 1.3.3. 2013. Available online: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.3.pdf (accessed on 27 Sept 2013)

[15] OpenFlow Switch Consortium and Others. OpenFlow Switch Specification Version 1.3.0. 2012. Available online: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf (accessed on 25 November 2014).

[16] F. Hu, ed. Network Innovation through OpenFlow and SDN: Principles and Design, CRC Press, Boca Raton, FL, 2014.

[17] Available online: www.iperf.fa