PANORAMA: Real-time Bird's Eye View of an OpenFlow Network

Ankit Gangwal, Mauro Conti
Department of Mathematics,
University of Padua, Italy.
Email: {ankit.gangwal, conti}@math.unipd.it

Manoj Singh Gaur
Department of Computer Science & Engineering,
Malaviya National Institute of Technology, India.
Email: gaurms@mnit.ac.in

***Abstract*—Software Defined Network (SDN) is an emerging networking paradigm that has gained enormous attention from the industries as well as the research community. SDN decouples data plane and control plane. The direct programmability of the control plane allows us to develop routing algorithms, which can accommodate versatile requirements of diverse network applications. On another side, Quality-of-Service provisioning, traffic engineering, etc., require accurate traffic measurements because an up-to-date view of the network facilitates service providers to optimize network performance. Existing approaches of traffic measurements demand either additional resources or alteration to infrastructure.**

In this paper, we present a collection of lightweight mechanisms for obtaining real-time network information in SDN environment. In particular, our mechanisms aim to obtain per-flow and per-port traffic statistics, topology information, data transfer rate for each network link, etc. Since our approach exploits the built-in capabilities of OpenFlow protocol, it does not require any changes to the infrastructure. We also implemented our proposed mechanisms and developed Panorama, a graphical user interface for real-time presentation of the obtained information.

***Index Terms*—SDN, OpenFlow, Real-time, Traffic Statistics, Network Monitoring**

I. INTRODUCTION

With increasing popularity of real-time services such as voice and video, network monitoring has become a significant task in network operation. In order to provide Quality-of-Service assurance for such services and operations like traffic engineering and network security require precise information about network "health". Due to the explosion of traffic volume in IP networks, it has become exceedingly difficult to acquire accurate traffic statistics. Network monitoring has been an active area of research. Current traffic estimation techniques such as flow-based measurements require several precious resources, e.g., processing power and bandwidth while other solutions compel expensive changes to the infrastructure. These limitations of currently available solutions raise demand for an efficient network management technique that is competent to furnish an accurate, precise and real-time view of the network while being inexpensive and simple to implement. In this paper, we propose integrated mechanisms for obtaining real-time network information that includes per-flow and per-port traffic statistics, topology information, data transfer rate for each network link, etc., for OpenFlow-based SDN environment. The collected information can be presented on our developed GUI, Panorama[1].

The remainder of this paper is organized as follows. Section II presents a brief summary of related work. Section III elaborates underlying architecture and working principles of our approach. The details of the prototype implementation and verification are discussed in Section IV. Finally, Section V concludes the paper and explores the future directions related to this work.

II. RELATED WORK

In SDN [1] environment, the key idea is to separate the data and control plane of network devices. The data plane includes several forwarding devices that provide the forwarding of packets. The data plane is controlled by the control plane. The control plane consists of at least one decision-making entity called the controller, which has a global view of the network of its domain. The controller communicates with the forwarding devices, and it decides the route of data packets in the network.

The controller and the switches communicate via OpenFlow [2] protocol. It creates a secure communication channel between the controller and the switches. The controller instructs the switches by simply updating their "flow table" via the OpenFlow protocol. Apart from adding flow table entries, the controller can also delete or modify existing entries. It can also collect logs, which are stored in the form of counters, from the forwarding devices. These logs can be used to calculate various traffic statistics. Utilizing topology information and the real-time traffic statistics, the controller can optimize the route for data packets. Hence, SDN allows network operators to develop application-specific route decision strategies considering the network topology information and the obtained traffic statistics.

Network traffic measurement approaches can be classified into two categories: active and passive. In active measurement approach, traffic flows are continuously monitored by injecting special probe packets. Although an active measurement of traffic flows can fulfill on-demand requests for traffic statistics, however, it involves a significant amount of overhead that may also interfere with critical traffic flows. On another side, in passive measurement approach, the network is probed at a

978-1-5090-4429-0/17/$31.00 ©2017 IEEE

[1]Panorama is available at `spritz.math.unipd.it/projects/pm/`

predefined interval with less overhead compared to the active measurement approach.

Jose et al. [3] have proposed an active measurement mechanism for measuring large-scale traffic aggregation by matching switch rules. However, it requires continuous updates to fetch the matching rules. OpenNetMon [4] adaptively fetches data from the switches where the accuracy increases at the expense of measuring overhead. iSTAMP [5] partitions Ternary Content Addressable Memory (TCAM) for aggregate and de-aggregate traffic. However, it requires an additional mechanism to "stamp" the flows. Zhang et al. [6] have proposed a prediction based algorithm to detect anomalies. However, the approach is restricted only to identified flows. OpenTM [7] constantly polls a switch for collecting flow statistics and produces high accuracy at the cost of high overhead of polling. Payless [8] employs an adaptive algorithm for polling flow statistics with both high- and low-frequency interval. In this approach, accuracy and overhead vary with the length of the polling interval. HONE [9] deploys software agents in every host and a module that interacts with network devices to periodically collect traffic data. The major hindrance in the deployment of this approach is the requirement of installing software agents in every host leading to scalability issues. PLANCK [10] utilizes port mirroring to obtain statistical data rapidly, but in a large network scenario, the traffic could exhaust the capacity of the ports. OpenSample [11] uses sFlow [12] and TCP sequence numbers for achieving high accuracy with low latency. FlowSense [13] uses PacketIn and FlowRemoved messages for low overhead passive measurement. The techniques discussed in [3–10] are active measurement approaches while techniques in [11, 13] are passive measurement approaches.

Existing network monitoring techniques require either expensive changes to the infrastructure or heavy computational resources. In this paper, we address the demand for accurate and precise network monitoring mechanisms, which are not only inexpensive but are easy to deploy.

III. Panorama

Panorama provides a real-time bird's eye view of an OpenFlow Network. Panorama collects and depicts vital information about the network. Panorama is developed as a POX [14] module to work along with any forwarding module, as shown in Figure 1. The forwarding module provides forwarding rules while Panorama collects and streams network information. The benefit of this design approach is that Panorama can also be used as a verification tool to verify the correctness of forwarding functionality developed by OpenFlow developers. The remainder of this section elaborates various type of network information collected by Panorama and their implementation details.

A. Network Configuration

Real-time network topology update is an essential requirement for estimating the "health" of the network. Network topology updates include liveliness report of switches, links, and hosts in the network.

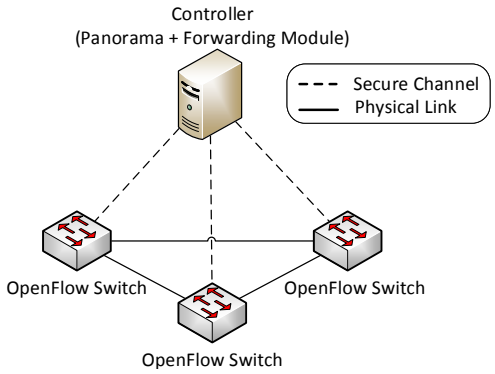

Figure 1: Co-existence of Panorama with forwarding module

1) Switch Discovery: A switch exchanges handshake messages when it connects to the controller. Upon successful connection, the controller learns that a switch with a unique identifier called "DataPath ID" (DPID) is connected to it. Information obtained for each discovered switch includes:

a Serial number.
b Software version.
c Vendor.
d Hardware type.
e DPID.

The connection between switch and controller terminates either because the switch was restarted or it has been turned-off. In such event, the controller learns that the switch has been disconnected.

2) Link Discovery: LLDP (Link Layer Discovery Protocol) is used to discover a switch-to-switch link. OpenFlow controller sends controller-specific LLDP packets to each discovered switch as PACKET_OUT messages. The controller also sends forwarding rule for such packets. The forwarding rule asks the switches to broadcast this packet with their DPID. When the broadcasted packet reaches an adjacent switch, it sends this packet to the controller asking for a forwarding rule, and the controller learns that there is an unidirectional link the two switches. The entire process is illustrated in Figure 2. Information stored for each discovered link includes:

a DPID of switches, between which a link exists.
b Port number of the switches, through which link is established.

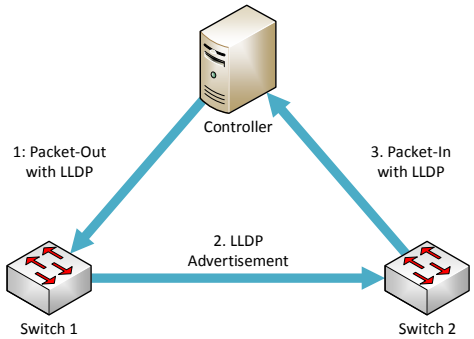

Figure 2: Link discovery in OpenFlow network

When a switch detects that a link to an adjacent switch has been removed or failed, it informs the controller by raising an event.

3) Host Discovery: When a flow of packets from a host reaches to a switch, the switch matches the packets with existing rules. For the first packet from a host, there would be no matching rule. Hence, the switch sends this packet to controller asking for a rule. By inspecting the packet, the controller learns the location of the host. Considering the facts that a host might move from one switch to another (as in the case of wireless connection) or DHCP might reassign IP addresses to the hosts, Panorama maintains a timer for each discovered hosts and a clean up is performed periodically. The duration of this period is usually greater than the expected hard timeouts because removal of a rule would force subsequent packet in the same flow from the host to flow towards the controller again. Information obtained from each discovered host includes:

a IP Address.
b MAC Address.
c Association Switch.
d Switch Port No.
e Time, which defines the time when the host was discovered.

B. Port Stats

OpenFlow PORT_STATS messages are used to obtain per-port statistics for each port on a switch. When a switch receives a PORT_STATS_REQUEST, it replies with a number of transmitted packets, transmitted bytes, transmit errors, packet dropped by TX, collisions, received packets, received bytes, receive errors, packet dropped by RX, packets with RX overrun, frame alignment errors and CRC errors for each of its port. Table I shows the main components of PORT_STAT_REPLY.

port_no	tx_packets	tx_bytes	tx_errors	tx_dropped	collisions	rx_packets	rx_bytes	rx_errors	rx_dropped	rx_over_err	rx_frame_err	rx_crc_err

Table I: Main components of port stats

C. Aggregate Stats

Aggregate statistics for a switch indicate the total number of currently installed flows, the total number of packets and the total number of bytes processed by the switch for these flows, as shown in Table II.

flow_count	packet_count	byte_count

Table II: Main components of aggregate stats

D. Flow Stats

OpenFlow FLOW_STATS messages are used to obtain per-flow statistics for every flow rule installed on a switch. A flow table consists of several flow entries, and each flow entry contains various fields as indicated in Table III.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Table III: Main components of a flow entry

1) **Match Fields:** These are the field against which packet headers are matched. A match field may be exact or may be wild-carded (match any value). Table IV shows the main components of the match fields.

Ingress Port	Ether Src.	Ether Dst.	Ether Type	VLAN Id	VLAN Priority	MPLS Label	MPLS Traffic Class	IP Src.	IP Dst.	IP Proto.	IP ToS	Transp. Src. Port	Transp. Dst. Port

Table IV: Main components of match fields

2) **Priority:** It is an integer number that determines the matching precedence of a flow entry. The higher the number, the higher the priority.
3) **Counters:** Counters are updated when packets are matched. Counters are maintained for each port, flow entry, flow table, etc.
4) **Instructions:** Instructions are executed when a packet matches a flow entry. Instruction may be "Required Instruction" or "Optional Instruction". Instructions either modify pipeline processing or contain a set of actions to be performed for the match.
5) **Timeouts:** A flow entry may have two type of timeouts: hard and idle. Timeouts specify either the maximum amount of time or an idle time before a switch evicts a flow.
6) **Cookies:** Controller usages cookies to filter flow modification, statistics, and deletion.

An OpenFlow switch should essentially have at least one flow table, and it can optionally have more than one flow tables as well. The advantage is that many network deployment demand orthogonal processing of packets (e.g., QoS, ACL, and routing). Using a single flow table to implement all of those processing requirements can create a massive rule-set in the flow table. Matching rules in a single large flow table can be time-consuming. Using multiple tables may properly decouple those processing requirements. Packet processing through multiple flow tables is called the OpenFlow pipeline processing. Panorama reports pipeline processing in the instructions field.

E. Link Data Transfer Rate

One of the important aspects of network monitoring is to estimate the data transfer rate (DTR) of the network links. Panorama calculates uni-directional and bi-directional DTR of a link between a pair of switches using the port statistics received from the switches, as shown in Figure 3. Calculation of uni-directional and bi-directional DTR in bytes per second (Bps) is shown in Eq. 1 and Eq. 2 respectively.

$$DTR^{Uni}_{S1 \rightarrow S2} = \frac{(Cur^{S1}_{tx\_bytes} - Pre^{S1}_{tx\_bytes})}{(Cur^{S1}_{time} - Pre^{S1}_{time})}. \quad (1)$$

Where $Cur^{S}_{tx\_bytes}$ denotes the number of transmitted bytes in the current observation at switch S, and $Pre^{S}_{tx\_bytes}$ refers

to the number of transmitted bytes in the previous observation at switch S. While Cur^{S}_{time} and Pre^{S}_{time} denotes the current time and previous time of observation at switch S.

$$DTR^{Bi}_{S1 \leftrightarrow S2} = \frac{(Cur^{S1}_{(tx+rx)\_bytes} - Pre^{S1}_{(tx+rx)\_bytes})}{(Cur^{S1}_{time} - Pre^{S1}_{time})}. \quad (2)$$

Where $Cur^{S}_{(tx+rx)\_bytes}$ is the total number of transmitted and received bytes in the current observation at switch S, and $Pre^{S}_{(tx+rx)\_bytes}$ is the total number of transmitted and received bytes in the previous observation at switch S.

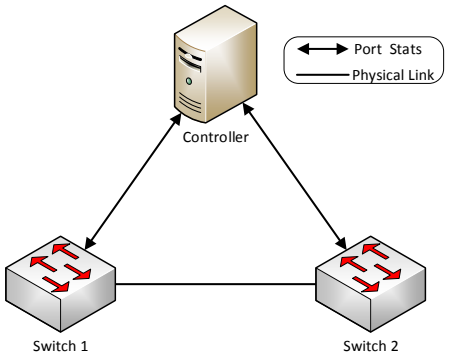

Figure 3: Measuring data transfer rate

F. Link Delay

Delay introduced by a link and link loss play a crucial role in Quality-of-Service provisioning. The work in [15] demonstrates mechanisms to measure link delay and loss. The controller injects probe packets to calculate delay on the links, as shown in Figure 4. At $t0$, it sends UDP packet to switch 1. The controller also installs a new rule instructing switch 1 to send this probe packet to switch 2. At $t1$ the packet flows from switch 1 to switch 2. Since a matching rule is not installed in switch 2, the switch sends the packet to the controller at $t2$. The controller receives this packet at $t3$. Since the controller maintains a constant connection with every switch, hence, it can estimate the delay between itself and the switches, as shown in Eq. 3. This enables the controller to estimate the delay between every pair of switches, as shown in Eq. 4.

$$t1 = t3 = 0.5 * (t_b - t_a). \quad (3)$$

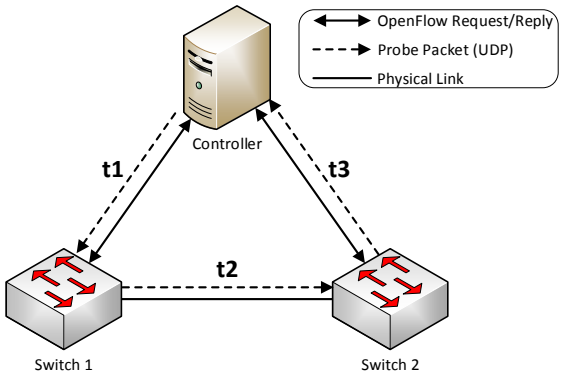

Figure 4: Measuring link delay

$$Time_{total} = t1 + t2 + t3 \Rightarrow t2 = Time_{total} - t1 - t3. \quad (4)$$

Where t_a is the time when an OpenFlow request message is sent, and t_b is the time when an OpenFlow reply message is received.

G. Link Loss

When a flow arrives at a switch and a matching rule does not exist in the switch, the first packet of the flow is sent towards the controller. The controller installs corresponding instruction in the flow table of the switches comprising the path chosen by the controller. As soon as the flow expires, the switch indicates this event to the controller. The entire process is illustrated in Figure 5. The flow is installed at time $t1$ using FLOW_MOD messages. After the flow rule expires at time $t2$ and $t3$, the controller receives FLOW_REMOVED messages from switch 1 and switch 2 respectively. Those messages include specific statistics for the flow such as the number of packets, bytes. The packet-loss rate ($Loss_{\%}$) can be measured on the basis of those statistics, as shown in Eq. 5.

$$Loss_{\%} = (1 - packets_{Switch2}/packets_{Switch1}) * 100. \quad (5)$$

Where $packets_{Switch1}$ is the number of packets transmitted from switch 1, and $packets_{Switch2}$ is the number of packets received at switch 2.

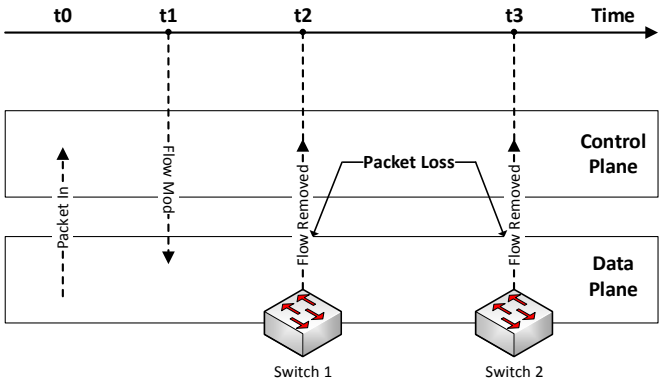

Figure 5: Measuring link loss

IV. Experiment and Verification

Panorama is tested on a physical testbed as well as in an emulation environment. The physical testbed comprises of an HP Aruba 5406R zl2 switch with Panorama running on a PC with 4th Gen Intel Core i3 processor with Intel HD Graphics, 4 GB of RAM, 500 GB HDD. Here, the physical switch is connected to two PC host.The emulation was done using Mininet[2] as the network emulator on a laptop with AMD A4-5000 1.5 GHz Quad Core APU with Radeon HD Graphics, 4 GB of RAM, 500 GB HDD. Here, Open vSwitch[3] serves as an OpenFlow-enabled switch. The emulated network topology is shown in Figure 6, here every edge switch is connected to three hosts. Iperf[4] is used to generate data traffic. To verify port, aggregate, and flow statistics `dpctl`[5] utility is used.

[2]Mininet - `http://mininet.org/`
[3]Open vSwitch - `http://openvswitch.org/`
[4]Iperf - `https://iperf.fr/`
[5]dpctl - `https://github.com/CPqD/ofsoftswitch13/wiki/`

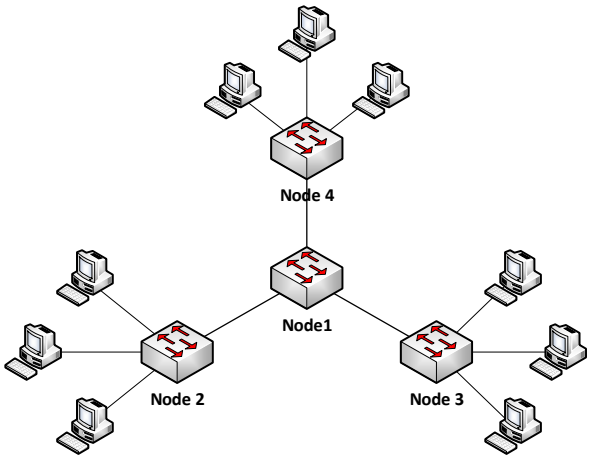

Figure 6: Evaluated network topology

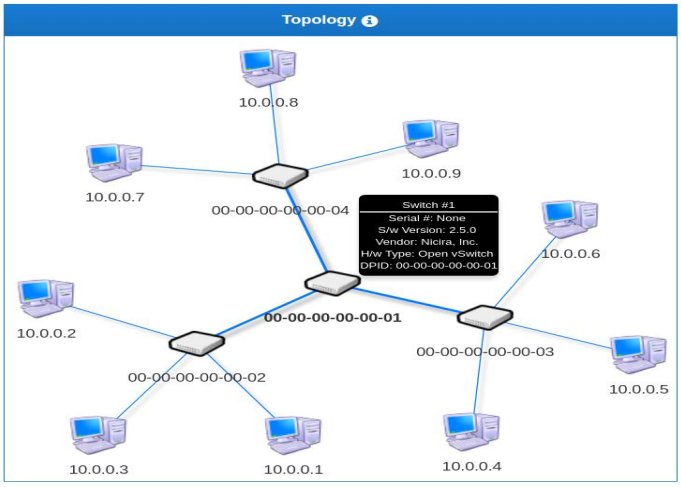

Figure 7: Discovered network topology

Figure 7 depicts the discovered network topology, Panorama labels each switch with its DPID and each host with its IP address. A comparison between Figure 6 and Figure 7 verifies the correctness of the discovered network topology. Due to space limitation, subsequent results are shown for only Node 1, which is referred as "s1" in the emulation. Figure 9 shows the computed port, aggregate, and flow statistics. The results can be verified by comparing Figure 9 with `dpctl` output for switch "s1", shown in Figure 8. Figure 10 illustrates the estimated data transfer rate of each link when each host pings every other host once.

Our proposed approach is lightweight as it relies on regularly exchanged OpenFlow messages. At the same time, it is inexpensive as it requires neither alteration to infrastructure nor injection of probe packets. On another side, it is also easy to deploy since it is a controller based measurement scheme.

```
mininet> dpctl dump-ports
*** s1 ------------------------------------------------------------------------
OFPST_PORT reply (xid=0x2): 4 ports
  port LOCAL: rx pkts=0, bytes=0, drop=125, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
  port  1: rx pkts=89, bytes=9071, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=154, bytes=17196, drop=0, errs=0, coll=0
  port  2: rx pkts=87, bytes=8940, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=153, bytes=16880, drop=0, errs=0, coll=0
  port  3: rx pkts=92, bytes=9317, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=154, bytes=17553, drop=0, errs=0, coll=0
```

(a) Port statistics

```
mininet> dpctl dump-aggregate
*** s1 ------------------------------------------------------------------------
NXST_AGGREGATE reply (xid=0x4): packet_count=166 byte_count=6806 flow_count=1
```

(b) Aggregate statistics

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=79.706s, table=0, n_packets=43, n_bytes=1763, idle_age=2,
priority=65000,dl_dst=01:23:20:00:00:01,dl_type=0x88cc actions=CONTROLLER:65535
```

(c) Flow statistics

Figure 8: `dpctl` output for switch "s1"

4 — 00-00-00-00-00-01

port_no	tx_packets	tx_bytes	tx_errors	tx_dropped	collisions	rx_packets	rx_bytes	rx_errors	rx_dropped	rx_over_err	rx_frame_err	rx_crc_err
65534	0	0	0	0	0	0	0	0	125	0	0	0
1	154	17196	0	0	0	89	9071	0	0	0	0	0
2	153	16880	0	0	0	87	8940	0	0	0	0	0
3	154	17553	0	0	0	92	9317	0	0	0	0	0

(a) Port statistics

1 — 00-00-00-00-00-01

flow_count	packet_count	byte_count
1	166	6806

(b) Aggregate statistics

4 — 00-00-00-00-00-01

match											actions	priority	timeout		duration		count		table_id	cookie
dl_src	dl_dst	dl_type	dl_vlan	nw_src	nw_dst	nw_proto	nw_tos	tp_src	tp_dst	in_port			idle_timeout	hard_timeout	duration_sec	duration_nsec	packet_count	byte_count		
*	01:23:20:00:00:01	LLDP	*	*	*	*	*	*	*	*	max_len : 65535 port : OFPP_CONTROLLER type : OFPAT_OUTPUT	65000	0	0	79	5.32e+8	43	1763	0	0

(c) Flow statistics

Figure 9: Computed statistics for switch "s1"

Uni-Directional (bps)				
Switch	00-00-00-00-00-01	00-00-00-00-00-02	00-00-00-00-00-03	00-00-00-00-00-04
00-00-00-00-00-01	N/A	8544.46	14003.40	12449.31
00-00-00-00-00-02	8600.66	N/A	N/A	N/A
00-00-00-00-00-03	12767.29	N/A	N/A	N/A
00-00-00-00-00-04	13469.73	N/A	N/A	N/A

(a) Uni-directional DTR

Bi-Directional (bps)				
Switch	00-00-00-00-00-01	00-00-00-00-00-02	00-00-00-00-00-03	00-00-00-00-00-04
00-00-00-00-00-01	N/A	17084.75	27222.60	25670.87
00-00-00-00-00-02	17197.32	N/A	N/A	N/A
00-00-00-00-00-03	25528.73	N/A	N/A	N/A
00-00-00-00-00-04	26140.47	N/A	N/A	N/A

(b) Bi-directional DTR

Figure 10: Estimated data transfer rate in response to `pingall` command

V. Conclusion and Future Work

In this paper, we have presented light-weight network monitoring mechanisms for SDN environment. These mechanisms employ the built-in capabilities of OpenFlow protocol. The obtained information can be presented in real-time on our developed GUI, Panorama. In future, we shall explore how other metrics such as congestion, jitter can be calculated in an OpenFlow-based SDN environment. We would also like to extend the GUI to manifest link loss and delay. We also hope to develop Panorama as a controller independent module.

VI. Acknowledgments

Mauro Conti is supported by a Marie Curie Fellowship funded by the European Commission (agreement PCIG11-GA-2012-321980). This work is also partially supported by the EU TagItSmart! Project (agreement H2020-ICT30-2015-688061), the EU-India REACH Project (agreement ICI+/2014/342-896). This work is partially supported by the grant n. 2017-166478 (3696) from Cisco University Research Program Fund and Silicon Valley Community Foundation.

References

[1] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 4, pp. 1955–1980, 2014.

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

[3] L. Jose, M. Yu, and J. Rexford, "Online Measurement of Large Traffic Aggregates on Commodity Switches," in *Hot-ICE*, 2011.

[4] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *IEEE NOMS*, 2014, pp. 1–8.

[5] M. Malboubi, L. Wang, C. Chuah, and P. Sharma, "Intelligent sdn based traffic (de) aggregation and measurement paradigm (istamp)," in *IEEE INFOCOM Conference on Computer Communications*, 2014, pp. 934–942.

[6] Y. Zhang, "An adaptive flow counting method for anomaly detection in SDN," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 25–30.

[7] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: traffic matrix estimator for OpenFlow networks," in *International Conference on Passive and Active Network Measurement*. Springer, 2010, pp. 201–210.

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

[9] P. Sun, M. Yu, M. J. Freedman, J. Rexford, and D. Walker, "Hone: Joint host-network traffic management in software-defined networks," *Journal of Network and Systems Management*, vol. 23, no. 2, pp. 374–399, 2015.

[10] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca, "Planck: Millisecond-scale monitoring and control for commodity networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 407–418, 2015.

[11] C. Dixon, W. Felter, and J. Carter, "OpenSample: A Low-latency, Sampling-based Measurement Platform for SDN," *IBM Research Division*, 2014.

[12] sFlow. http://www.sflow.org/.

[13] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in *International Conference on Passive and Active Network Measurement*. Springer, 2013, pp. 31–41.

[14] POX. http://github.com/noxrepo/pox/.

[15] V. N. Gourov, "Network Monitoring with Software Defined Networking: Towards OpenFlow network monitoring," Ph.D. dissertation, TU Delft, Delft University of Technology, 2013.