Real Time Monitoring of Packet Loss in Software Defined Networks

Yash Sinha[1(✉)], Shikhar Vashishth[2], and K. Haribabu[1]

[1] Department of Computer Science and Information Systems, BITS, Pilani, Pilani Campus, Pilani, India
h2016077@pilani.bits-pilani.ac.in

[2] Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India

Abstract. In order to meet QoS demands from customers, currently, ISPs over-provision capacity. Networks need to continuously monitor performance metrics, such as bandwidth, packet loss etc., in order to quickly adapt forwarding rules in response to changes in the workload. The packet loss metric is also required by network administrators and ISPs to identify clusters in network that are vulnerable to congestion. However, the existing solutions either require special instrumentation of the network or impose significant measurement overhead.

Software-Defined Networking (SDN), an emerging paradigm in networking advocates separation of the data plane and the control plane, separating the network's control logic from the underlying routers and switches, leaving a logically centralized software program to control the behavior of the entire network, and introducing network programmability. Further, OpenFlow allows to implement fine-grained Traffic Engineering (TE) and provides flexibility to determine and enforce end-to-end QoS parameters.

In this paper, we present an approach for monitoring and measuring online per-flow as well as per-port packet loss statistics in SDN. The controller polls all the switches of the network periodically for port and flow statistics via OpenFlow 1.3 multipart messages. The OpenFlow compliant switches send cumulative statistics of sent and received packets to the controller that includes raw packets (control, non-user generated packets responsible for network management); which, although not being part of the end-to-end data traffic, get counted and act as noise in the statistics. The proposed method takes into account the effect of raw packets and thus, hamper the accuracy of methods.

Other implementations propose approaches for per-flow packet loss only. We also take into account the effect of raw packets (control, non-user generated packets) which makes our packet loss estimation more accurate than other implementations. We also present a study of extrapolation techniques for predicting packet loss within poll interval.

Keywords: Software Defined Networks · Packet loss · OpenFlow
Quality of service (QoS) · Traffic Engineering (TE)

1 Introduction

Software-Defined Networking (SDN) along with OpenFlow [1] has inspired both academia and industry to test new ideas in fields of customized architecture, different algorithms, novel protocols, especially network status monitoring making enhanced Quality of service (QoS) possible.

Packet loss is one of the significant performance metrics used to determine QoS and in network diagnostics. Packet loss due to congestion is a fundamental issue in modern day networks for both network researchers and network operators.

There are various approaches to monitoring the network. The two common approaches are the passive and active approaches. The passive approaches do not increase the traffic on the network for the measurements, but rely on installation of devices that watch the traffic as it passes by. This may require large investments. Flowsense [2] proposes a passive push based monitoring method using control messages sent by switches to the controller to estimate network metrics.

The active approach injects test packets into the network or sends packets to servers and applications to monitor the network. The benefit is that it can be run from virtually anywhere in the network and it gives an end to end perspective of the network behaviour; although it introduces additional network load which affects the network and therefore influences the accuracy of the measurements.

In this paper, we present an approach for monitoring and measuring online per-flow and per-port packet loss and its comparison with some of the novel methods for measuring packet loss proposed for SDN and OpenFlow [1] networks. Other implementations propose approaches for per-flow packet loss only and not for per-port packet loss. This can be utilized by network administrators and ISPs to identify clusters in large network that are vulnerable to congestion.

We compare the accuracy for TCP & UDP packets and for networks having single and multiple flows. We discuss why polling switches for flow and port statistics in linear way is better than other ways such as round robin, last switch, per-flow etc. We also explore how accurate extrapolation techniques are, for predicting packet loss within poll interval.

Rest of the paper is divided into Related Work, Network Model and Architecture, Proposed Approach, Experimental Setups, Results, Future Work and Conclusion.

2 Background and Related Work

Flowsense [2] uses control messages sent by switches to the controller to estimate network metrics. But its estimation is far from the actual value and it works well only when there is a large number of small duration flows. It cannot capture traffic bursts if they do not coincide with another flow's expiry.

OpenNetMon [3] is a OpenFlow controller module that monitors per-flow QoS metrics such as latency, throughput, delay and packet loss by polling flow source and destination switches. Polling is done for every path between every node pairs to be monitored and it is adaptively changed based on new flows' arrivals. But the controller does not know to find new paths based on real-time data. Since the monitoring targets are only limited to edge switches, it is difficult to obtain detailed flow statistics on other switches.

OpenTM [4] presents a heuristics-based monitoring method wherein it uses the routing information learned from the OpenFlow controller to intelligently choose the switches from which to obtain flow statistics, thus reducing the load on switching elements and improving the monitoring accuracy. However, constant polling involves considerable overhead.

OpenSketch [5], proposes a new SDN based monitoring architecture and a new OSDN protocol. This, however, requires an upgrade or replacement of all network nodes. Furthermore, standardization of a new protocol has been a long and tedious task.

PayLess [6] is a query-based monitoring framework for SDN which performs highly accurate information gathering in real-time without incurring significant network overhead. To achieve this goal, instead of letting controller continuously polling switches, an adaptive scheduling algorithm for polling is proposed to achieve the same level of accuracy as continuous polling with much less communication overhead.

3 Network Model and Architecture

First of all we explain the basic network model and the architecture used. The network is emulated using MiniNet Network Emulator [7] which runs a collection of virtual end-hosts, switches, routers, and links on a single Linux kernel. Furthermore, it allows us to create many custom topologies and emulate some link parameters like a real Ethernet interface, e.g., link speed, packet loss, and delay.

We use SDN enabled (i.e. OpenFlow [1] compliant) OpenVSwitch [8] switches and Ryu [9] controller to handle their control plane. We emulate the network topology using L2 learning switches wherein the switches examine each packet and learn the source-port mapping. Thereafter, the source MAC address gets associated with the port. If the destination of the packet is already associated with some port, the packet is sent to the given port, else it is be flooded on all ports of the switch.

Our code sits at the controller and computes the packet loss for the network. We use the controller to request the switches for port and flow statistics via the OpenFlow protocol 1.3. This is done periodically every k seconds by issuing PortStatsRequest and FlowStatsRequest requests. So network overhead involved is 2n messages, where n is the number of switches in the network for every k seconds.

We use tc tool of queueing discipline (also called qdisc) part of Ubuntu kernel's traffic control module to measure actual packet loss from switches emulated using MiniNet. MiniNet also uses qdisc internally to emulate link's properties such as bandwidth, delay etc. and manipulate other traffic settings. Therefore, qdisc can provide true packet loss statistics that can be used for comparison.

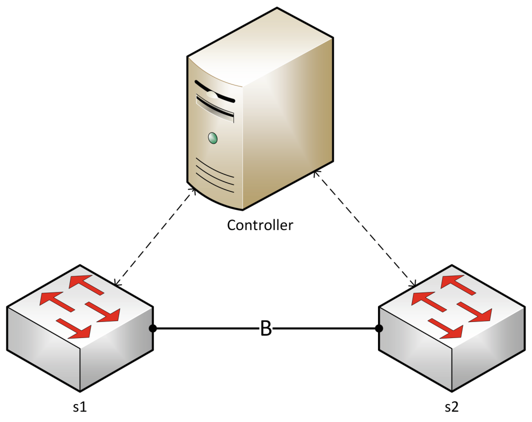


Fig. 1. Base model

We use iperf2 [10] which is a traffic generation tool that allows the user to experiment with different TCP and UDP parameters to see how they affect network performance. We also use Distributed Internet Traffic generator D-ITG [11] to generate bursty traffic to test extrapolation techniques.

We define the base model of our system consisting of two switches s1 and s2, connected by a link of bandwidth B. There is a direct logical link from each switch to the controller Cn. We identify a flow by 4-tuple: in port, out port, source MAC address and destination MAC address (Fig. 1).

Our model network consists of multiple such switches connected together in a various topologies with the link distances equal for all. The end switches are connected to hosts. We assume the switches and the hosts to be homogenous. One of the hosts acts as an iperf server and the other as iperf client, while running the experiment. The bandwidths for the links are variable which is why packet loss is expected to happen at some links in the network.

4 Proposed Approach

In this section we present our approach to estimate online packet loss, which is divided into following stages: 1. estimation of raw packets, 2. gathering port and flow statistics, 3. real time estimation, 4. extrapolation within poll interval.

4.1 Estimation of Raw Packets

We define raw packets as the packets exchanged in the network which are not a part of the active traffic but are used for network management such as for network discovery, multicast listeners discovery, requesting networking parameters from DHCP servers etc. These packets are necessary for many network protocols such as MDNS, NDP, MLD, DHCP etc.

The raw packets are absorbed in the intermediate switches and they are not meant for hosts. The SDN enabled switches send cumulative statistics of sent and received packets to the controller that includes raw packets. Therefore for calculating packet loss accurately, it is necessary to separate data packets from raw packets.

Using Emulation. We emulate the network with no active traffic between hosts for some time and the controller polls the switches queries the switches to accumulate the statistics about the packets exchanged. The network is called raw network. Later this data is used to calculate packet loss.

Using Mathematical Model. We emulated different network topologies for accumulating raw packet statistics and noticed a pattern in which packets are exchanged which was periodic and additive based on the topology of network.

We devised our own algorithm [12] for predicting raw packet flow in any network. The algorithm uses the spanning tree information about network topology and calculates the number of hosts and switches in the subnetwork for each interface of every switch in the network and based on that count, the raw packets are estimated.

4.2 Gathering Port and Flow Statistics

Port Statistics. Other implementations propose approaches for per-flow packet loss only and not for per-port packet loss. Even though OpenTM [4] advocates polling each path's switch randomly or in round robin as most efficient, and OpenNetMon polls each flow-path's last switch; we poll every switch of the network for port statistics.

The reasons are manifold. Firstly, collecting flow based statistics becomes complex and tedious in large networks with multiple flows. Further, non-edge switches generally have a large number of flows to maintain, making the query for flow statistics more expensive. When more flows exist, non-edge switches will be polled more frequently degrading efficiency. For a network with n switches there can be nC2 flows in the worst case. In contrast, in our approach there are at most n port statistics requests and size of query response and polling frequency is independent of number of flows in the network.

Therefore, our approach introduces relatively less additional network load which affects the network and influences the accuracy of the measurements.

Flow Statistics. For flow based statistics collection, the round robin switch selection becomes more complex in larger networks with multiple flows. Therefore, in our implementation we query the switches linearly that does not explode to n_2^C in the worst case.

4.3 Real Time Estimation

For calculating per port packet loss statistics at a given port, we subtract the RX packets (packets received on the port) at the destination port of the destination switch from TX packets (transmitted out of the port) packets at the source port of the source switch. Further, from this value we subtract the raw RX packets at the destination port of the destination switch from raw TX packets at the source port of the source switch.

$$packetLoss(perport) = [TX_a - RX_a] - [X_r - RX_r];$$

where, a: active network, r: raw network.

For calculating per flow statistics, we subtract the packet count at the destination port of the destination switch from the source port of the source switch

$$packetLoss(perflow) = Count_s - Count_d;$$

here, s: source, d: destination.

The raw packet correction cannot be applied to flows because no flows are generated in raw network and hence no data is collected.

4.4 Extrapolation within Poll Interval

Once the controller gets data, it uses two methodologies to predict data: extrapolation based on rate of packet loss in last k seconds; and extrapolation based on rate of packet loss and change in rate of packet loss in last k seconds.

5 Experimental Setups

5.1 Experimental Setup 1: Single Flow

For initial comparison purposes with OpenNetMon, a linear network topology consisting of three connected switches was used, with ends connected with two host systems (Fig. 2).

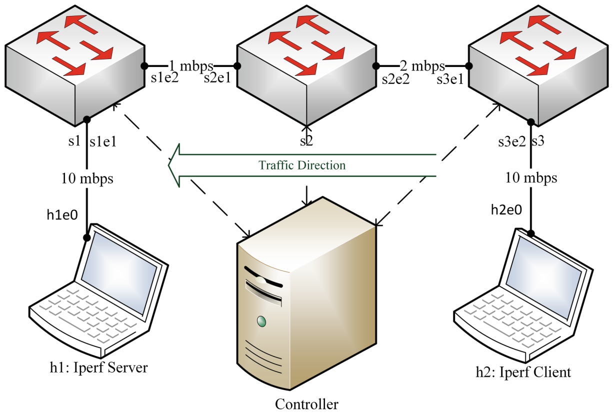


Fig. 2. Experimental setup for single flow

Here, $h\alpha e\beta$ denotes port of host and $s\alpha e\beta$ denotes port of switch, having links with bandwidths 10, 1, 2 and 10 mbps respectively. Iperf was used to send udp/tcp data from $h2$ to $h1$ at 5 mbps, this setting was designed so that approximately 1 mbps loss would occur at link2 and some loss would occur at link 3 as well. The polling frequency (k) was kept as 5 s.

5.2 Experimental Setup 2: Multiple Flows

(See Fig. 3).

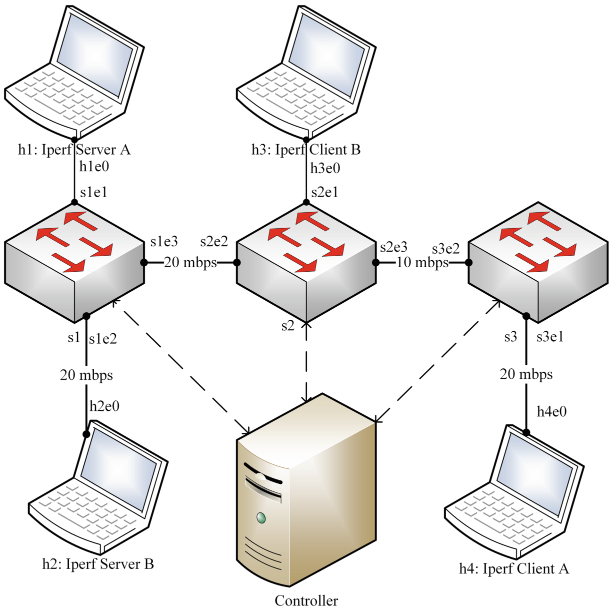


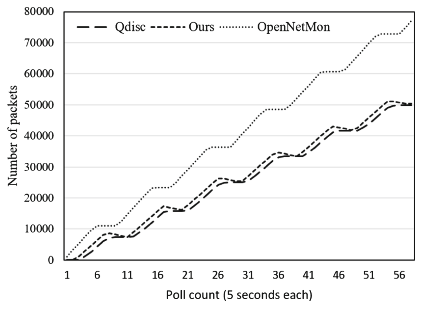
Fig. 3. Experimental setup for multiple flows

A linear network topology consisting of 3 connected switches is used. Switch $s1$ is connected to two host systems and one host each is connected at s2 and s3. Iperf UDP traffic of 6 mbps and 8 mbps is generated for h1–h4 and h2–h3 pair of hosts respectively.

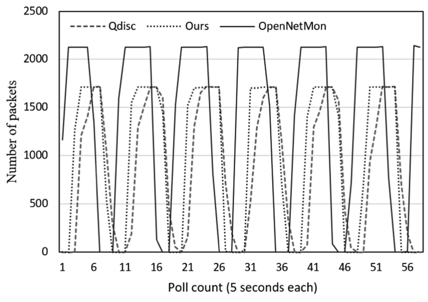
6 Results

6.1 Comparison of Accuracy for Single Flow

We ran experiments on the aforementioned topology for networks transmitting TCP and UDP traffic. On the same network we also ran the OpenNetMon module to compare the accuracy reported. Since there is only one flow, we assume per port loss to be equal to per flow loss, hence comparison is possible. We recorded the actual packet loss data by periodically calling qdisc's tc tool.



(a) Cumulative Packet Loss

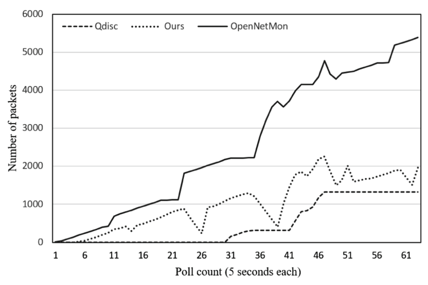


(b) Real time Packet Loss

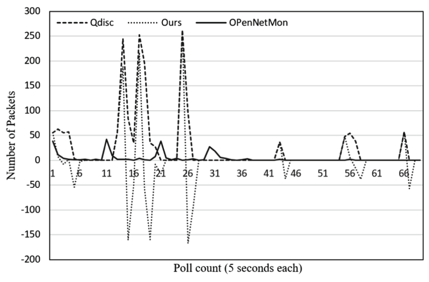
Fig. 4. UDP traffic

UDP Traffic. UDP traffic of bandwidth 5 mbps is generated using iperf from host h2 to host h1. As the graph shows, our proposed method closely matches with the packet loss reported by qdisc, whereas OpenNetMon reports lags behind as raw packets which are absorbed in the intermediate switches are reported as lost. We present both real time and cumulative statistics here (Fig. 4).

TCP Traffic. TCP traffic is generated using iperf from host h2 to host h1. As the graph shows, our proposed method matches with the packet loss reported by qdisc, whereas OpenNetMon reports lags behind as raw packets which are absorbed in the intermediate switches are reported as lost. We present both realtime and cumulative statistics here (Fig. 5).



(a) Cumulative Packet Loss



(b) Real time Packet Loss

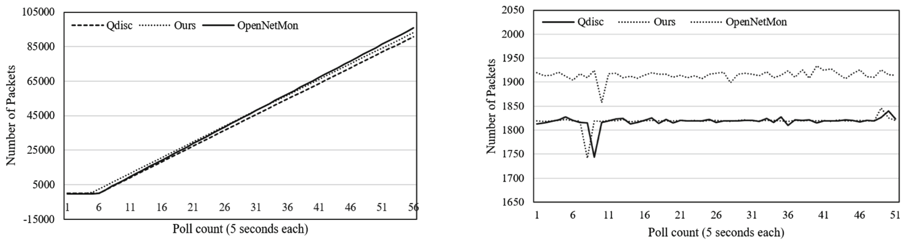
Fig. 5. TCP traffic

As compared to OpenNetMon, this method increases the accuracy of reported real-time packet loss (error is reduced to 2–3% from 9–10%) for aforementioned network.

6.2 Comparison of Accuracy for Multiple Flows

We present the comparison of total packet loss of two flows from h4 to h1 and h3 to h2 calculated by OpenNetMon, qdisc and our approach. As explained in Sect. 4.3, raw packet correction cannot be applied to flows; hence our approach and OpenNetMon's approach gives us identical results. We have skipped those plots.

Since qdisc reports statistics port wise only, for comparison we compare the total statistics of flow 1 and flow 2 combined. The raw packet correction has been applied to total packet loss. Clearly, we can see that OpenNetMon fails to account for the raw packets and thus reports packet loss erroneously (Fig. 6).



(a) Cumulative Packet Loss (b) Real time Packet Loss

Fig. 6. Multiple flows

6.3 Extrapolation Within Poll Interval

(See Fig. 7).

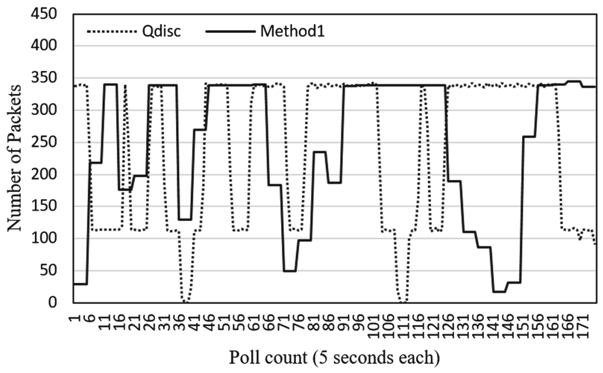


Fig. 7. Packet loss rate based extrapolation

Extrapolation Based on Rate of Packet Loss. For the poll interval we extrapolate the packet loss rate observed during the last 5 s i.e., the packet loss rate is assumed to sustain for the next five seconds. So accuracy largely depends on how dynamically the network traffic is changing and how frequently the controller is polling the switches for the statistics. For relatively stable network, extrapolation gives packet loss with good accuracy but if the network changes drastically within the poll interval, the controller and hence the extrapolation fails to capture it (Fig. 8).

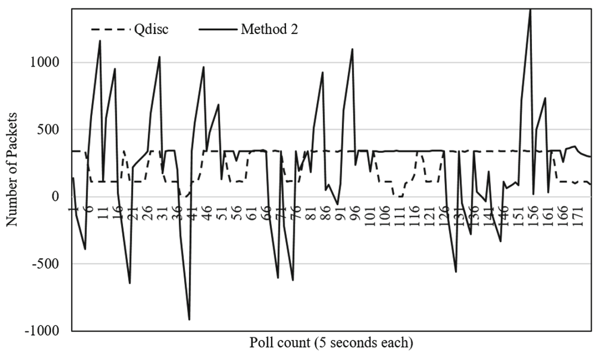


Fig. 8. Packet loss rate based extrapolation and change in packet loss rate

Packet Loss Rate Based Extrapolation and Change in Packet Loss Rate. Here, we extrapolate based on the packet loss rate and also the change in rate of packet loss. The extrapolation is highly sensitive to small changes in rate of change of packet loss, so this proves that the prediction model is not dependent on higher derivatives of time.

7 Future Work

As proposed by PayLess [6], we can reduce the network overhead by using an adaptive scheduling algorithm for polling. For reporting packet loss during the poll interval, we can use history based prediction. More features based on network topology can be added.

8 Conclusion

Network Admins and Internet service providers can utilize per port loss in SDN for identifying clusters in large networks which are congestion vulnerable. By taking into account the packet loss information the proposed method drastically increases the preciseness of obtained real-time packet loss statistics. It is also scalable to large dense networks, as it avoids polling edge switches on per flow basis.

References

1. McKeown, N., et al.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)
2. Yu, C., Lumezanu, C., Zhang, Y., Singh, V., Jiang, G., Madhyastha, H.V.: FlowSense: monitoring network utilization with zero measurement cost. In: Roughan, M., Chang, R. (eds.) PAM 2013. LNCS, vol. 7799, pp. 31–41. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36516-4_4
3. Van Adrichem, N.L.M., Doerr, C., Kuipers, F.: Opennetmon: network monitoring in openflow software-defined networks. In: 2014 IEEE Network Operations and Management Symposium (NOMS), pp. 1–8, 5 May 2014
4. Tootoonchian, A., Ghobadi, M., Ganjali, Y.: OpenTM: traffic matrix estimator for OpenFlow networks. In: Krishnamurthy, A., Plattner, B. (eds.) PAM 2010. LNCS, vol. 6032, pp. 201–210. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12334-4_21
5. Wellem, T., Lai, Y.-K., Chung, W.-Y.: A software defined sketch system for traffic monitoring. In: Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems, pp. 197–198, 7 May 2015
6. Chowdhury, S.R., et al.: Payless: a low cost network monitoring framework for software defined networks. In: 2014 IEEE Network Operations and Management Symposium (NOMS), pp. 1–9, 5 May 2014
7. Lantz, B., Heller, B., McKeown, N.: A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, p. 19, 20 October 2010
8. Open vSwitch (2015). http://openvswitch.org/download/. Accessed 15 Dec 2015
9. Ryu SDN Framework (2015). http://osrg.github.io/ryu. Accessed 15 Dec 2015
10. Tirumala, A., et al.: iPerf: the TCP/UDP bandwidth measurement tool. https://iperf.fr/
11. Avallone, S., et al.: D-ITG distributed internet traffic generator. In: Proceedings First International Conference on the Quantitative Evaluation of Systems, QEST 2004. IEEE (2004)
12. Sinha, Y., Vashishth, S., Haribabu, K.: Meticulous measurement of control packets in SDN. In: Proceedings of the Symposium on SDN Research. ACM (2017)