Quality-of-Service (QoS) Esitmation in Software-Defined Networks

Computer Networks II: Project Final Report

Xin Liu

Department of Computer Science,
Illinois Institute of Technology,
xliu125@hawk.iit.edu

1 Introduction

Multimedia applications have strict requirements of Quality- of-Service (QoS) in order to guarantee user experience. Thus, accurately measure the QoS metrics, such as throughput and latency, is a critical task. However, packet-switching network has the nature of multiplexed flows and best-effort forwarding, which makes the task difficult to accomplish. Software-Defined Networking (SDN) (*1*) is a newly proposed network architecture that separates the packet forwarding (data plane) and control logic (control plane). The forward- ing devices (i.e., switches) in SDN network keep simple data structures and process schemes locally, while the centralized entity (i.e., controller), which connects to all the forwarding devices, maintains network states and execute complicated control algorithm to operate the network. In addition to modifying the table entries on the switches, controller has the ability to pull the statistical information maintained by all the switches, which enables us to utilize these information to estimate the throughput of a flow. In this work, we use a simple formula to calculate the throughput, and conduct experiments to evaluate whether this calculation is close to the actual end-to-end throughput.

By using network emulation and Linux iperf test, we verified that the estimated result is relatively accurate in a simple linear topology, and the accuracy is related to which switch is chosen to monitor. From the result, we find there is a potential research direction on how to choose the target switches in a large-scale complicated network.

The rest of this report is organized as follows: section 2 gives a brief background on SDN and related work; section 3 shows the simple formula we use to esitmate the flow throughput; the emulation tool and throughput measurement tool is introduced at section 4, followed by the desciption of experiment setup, then the result analysis; finally section 5 gives the conclusion of this study and propose a future work.

2 Background and Related Work

Software-Defined Network (SDN) is a network architecture composed of three layers: Infraustructure layer, Control layer, and Application layer. As shown in Figure 1, the lowest layer is infrastructure layer which is composed of hardware devices with simple forwarding tables and match-action packet processing mechanism. The middle layer, controller layer, contains a (logically) central entity (i.e., controller) that communicate with the low-level hardware and provide services to the upper applications. The ”services” include topology information, flow statistics, shortest paths, etc. Network operators utilize these services to implement their network policies, and run as network applications at the top layer. While the communication protocol between Control layer and Application layer (”northbound API”) has not been standardized yet, there is a widely adopted protocol between Control layer and Infrastructure layer (”southbound API”), named OpenFlow (*2*). OpenFlow-enabled devices contain a pipeline of forwarding tables, each of which contains a number of table entries shown in Figure 2. The match field contains a set of values to identify the incoming packets to process, which means to apply the action specified by the action field. Meanwhile, for every packet that is matched by this entry, the corresponding

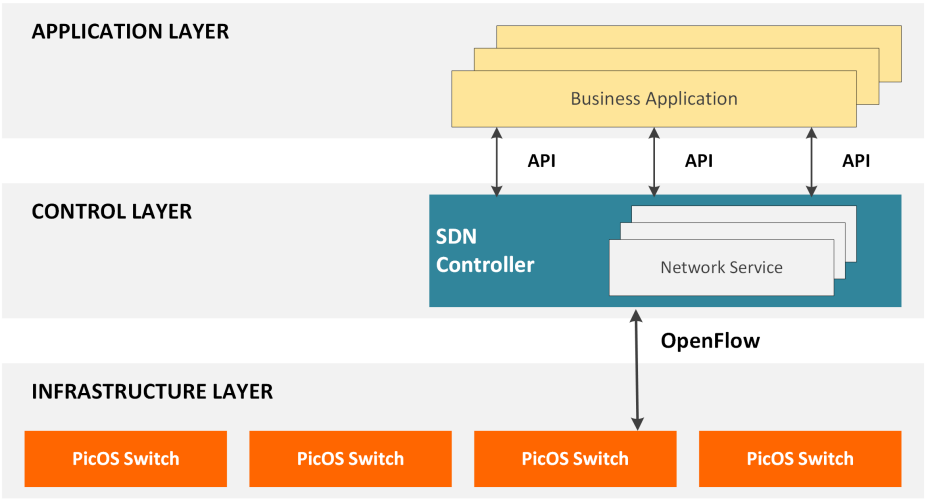


Figure 1: SDN Architecture

stats field is going to be updated. The controller is able to install/remove/update the entries on all the switches using the OpenFlow protocol, while the switches can notify the controller about certain events, such as the arrival of new (unmatched) packet, link failure, etc.

Another mechanism we are interested in here is that the controller pulls every entry's statistics periodically to monitor the network. Using these information, a controller can estimate the throughput of each flow and the remaining bandwidth of certain links, which provides the information for scheduling of future flows and/or re-routing existing paths to guarantee QoS. (*3*) is an example to illustrate this idea, and (*4*) implemented an application to further control the monitoring process. In this work, we use emulation to simply demonstrate the concept of end-to-end throughput estimation in SDN, and show the different results when monitoring different switches in a topology.

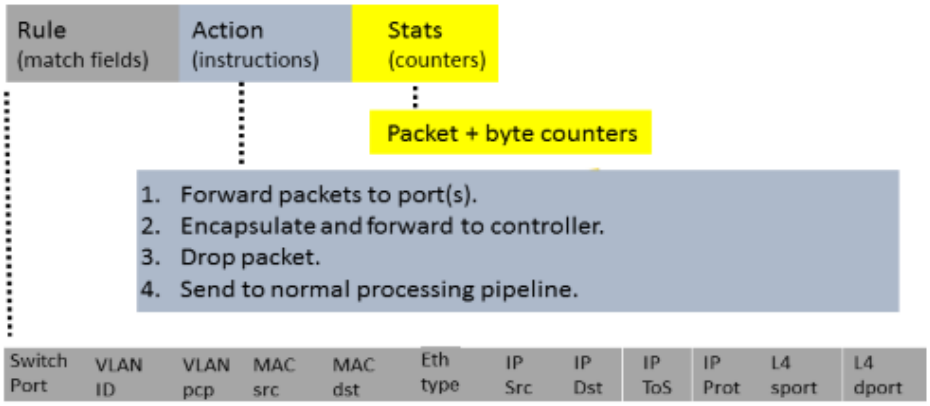


Figure 2: OpenFlow Table Entry

3 Estimating End-to-End Throughput

In this work, we use the simplest way to estimate the flow throughput, which is to pull the "number of bytes" processed by the entry and divide the elapsed time that this entry has been counting. Figure 3 shows a sample web GUI of a running controller, which displays the network topology running under it. Figure 4 shows the same controller's view of the table entries of the leftmost switch in the topology above. The right most two columns shows the number of packets and bytes processed by each table entry. For example, say flow A's entry has the "BYTES" column to be X, and this entry has been installed for t seconds, then we can estimate the flow A's throughput is

$$\frac{X \times 8}{t} \quad bits/sec \tag{1}$$

4 Evaluating Estimation Accuracy Using Emulation

To show how the esitmation works in a real network environment, we use a light-weight network emulator called Mininet (*5*) to generate a simple SDN network and run iperf test on the host pairs to compare the measured throughput and estimated result.

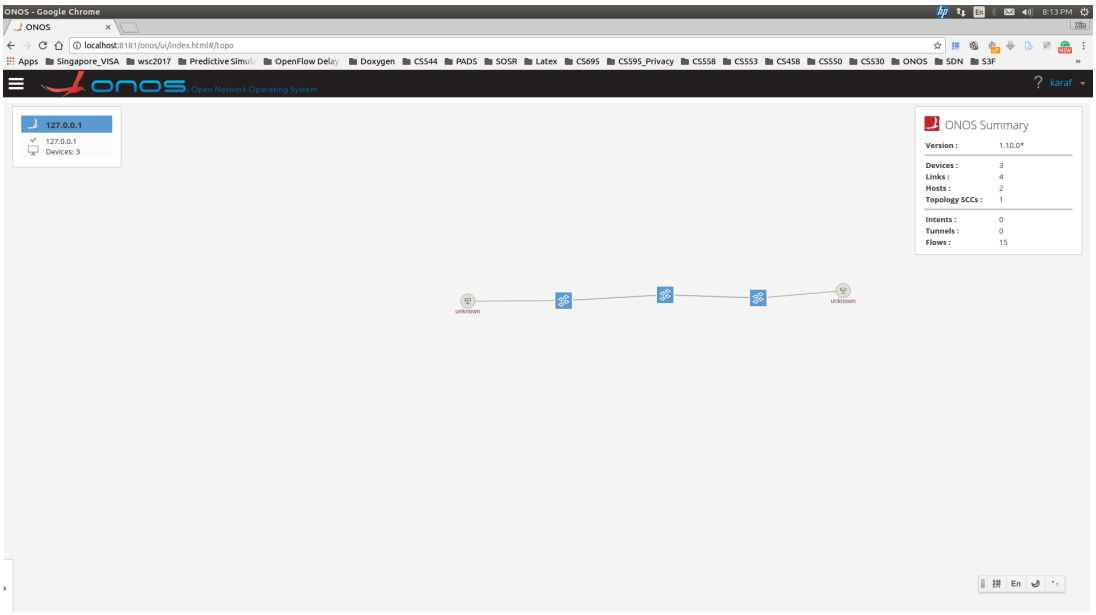


Figure 3: GUI of A Controller Showing Topology

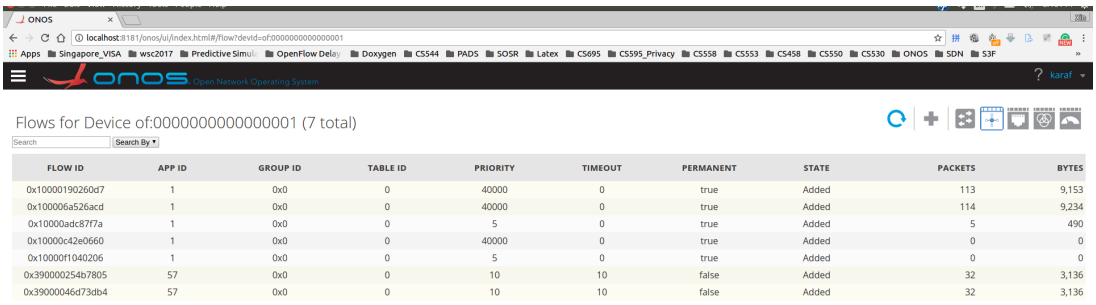


Figure 4: GUI of A Controller Showing Table Entries

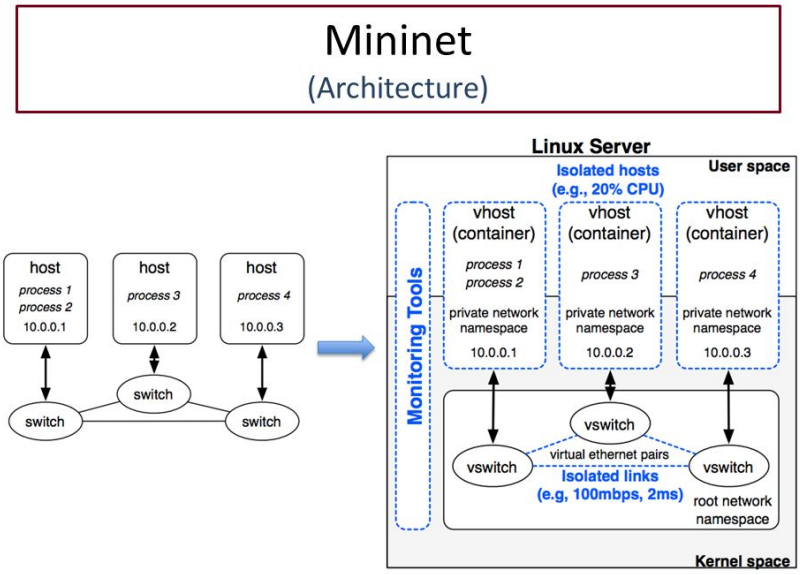


Figure 5: Mininet Architecture

4.1 Mininet Emulator

Mininet emulator use a virtualization technology called *container-based virtualization*, where all the "virtual machines" share the same Linux kernel and have their own namespaces from the application view. In Mininet, only the network namespace (i.e., network interface, eth0) is independent by each VM. As shown in Figure 5, the target network is shown on the left hand side, where three hosts is connected by three switches. In Mininet, these three hosts are simply three Linux processes, each with a private network namespace. The switches are software switches called OpenVSwitch, and the links are configured using Linux *tc* command, which contains different traffic shaping mechanism (e.g., token bucket) to guarrantee certain bandwidth and latency.

Mininet provide a Python API to users, who could write scripts to generate a network and trigger the virtual hosts to execute different programs (e.g., iperf client/server program). For example, Figure 6 shows a code snippet of generating a simple linear topology, which will be

```
class SimpleTopo(Topo):

    switchList = []

    host_1 = ""
    host_2 = ""

    length = 0

    def __init__(self, l):

        Topo.__init__(self)
        self.length = int(l)

        for i in xrange(self.length):
            self.switchList.append(self.addSwitch('s'+str(i+1)))

        for i in xrange(self.length-1):
            self.addLink(self.switchList[i], self.switchList[i+1], bw=1, use_tbf=True)

        self.host_1 = self.addHost('h1', ip='10.0.0.1')
        self.host_2 = self.addHost('h2', ip='10.0.0.2')
        self.addLink(self.host_1, self.switchList[0], bw=10, use_tbf=True)
        self.addLink(self.host_2, self.switchList[self.length-1], bw=10, use_tbf=True)
```

Figure 6: Code Snippet

introduced in the experiment subsection.

4.2 Ipef Throughput Test

Iperf (*6*) is an active throughput measurement tool to evaluate the throughput between two end-hosts along a path. To begin the test, one end-host act as the server by executing "iperf -s" command, and the other execute "iperf -c" to generate traffic and send to the server side. There are multiple parameters can be configured, such as protocols (TCP/UDP), running time, total bytes to send, etc.

4.3 Experiment Setup

The experiments has three scenarios. All of them follows a linear topology, where switches form a sequence and hosts reside on the two ends. As shown in Figure 7, the first one is a simple linear topology of three switches, with two end-host as client/server pair. To compare with the throughput measured at the host using iperf, we monitor the flow statistics on the ingress switch and egress switch, and calculate the esitmated throughput as stated in section 3.

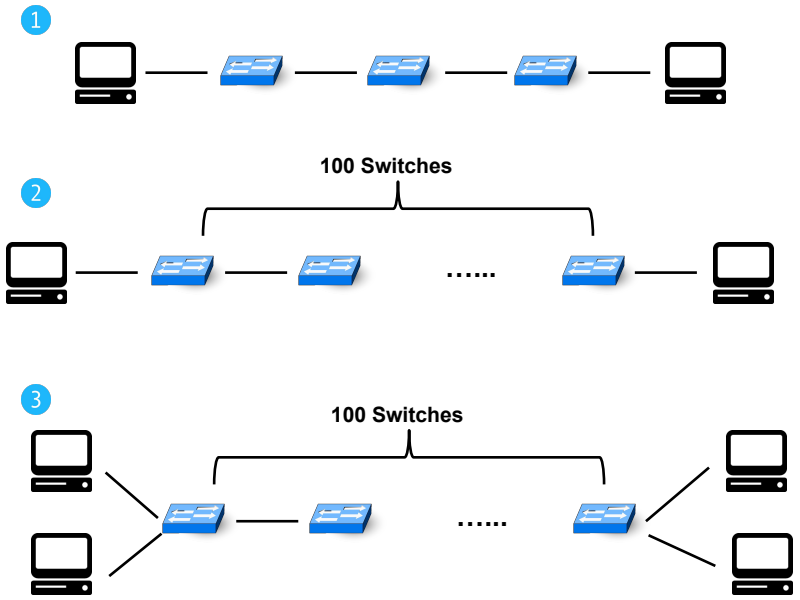


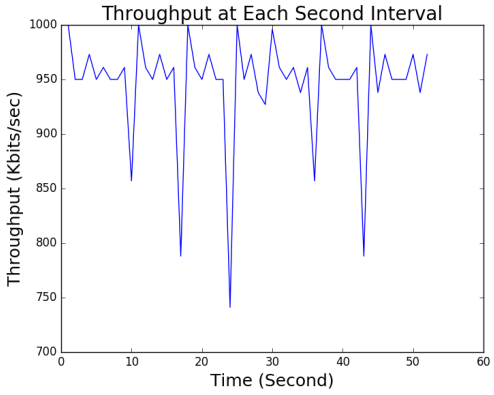
Figure 7: Three Scenarios

The second scenario extend the linear topology to 100 switches, and also measure the ingress and egress switches. At last, the third scenario has the same topology as the second one, but add another pair of hosts running the iperf program at the same time, to create centention of the bandwidth. We still measure the throughput of one flow, and collect the ingress/egress switches **corresponding flow**'s stats (there are two flows at all the switches).

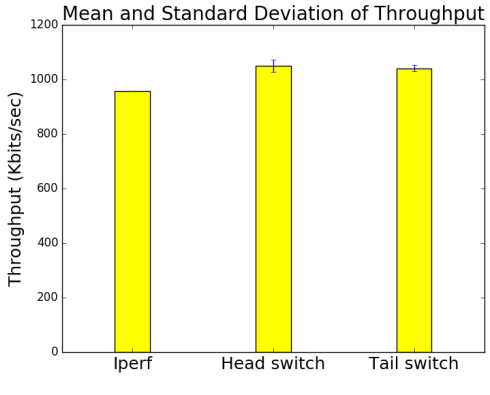
In all the topologies, the every link bandwidth is set 1 Mbps, and latency is set 1 msec. Each experiment is repeated 10 times.

4.4 Experiment Results and Analysis

Following the three scenarios, we have corresponding experiment results, which are shown in Figure 8, Figure 9, and Figure 10 accordingly. Each scenario's results are illustrated in the form of (a) iperf throughput measurement at each one-second time interval, of one trial; (b) the average of measured or calculated throughput over 10 repeatitions. The "head switch" means



(a) Iperf Result Timeline



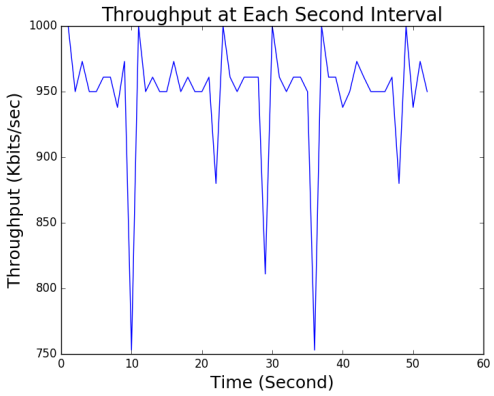
(b) Mean Throughput

Figure 8: Result of Linear-3 Topology

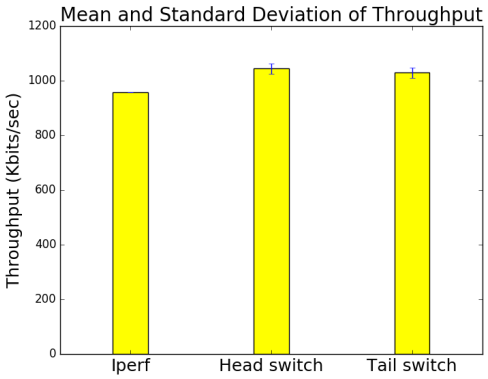
the throughput esitmated according to the stats on the ingress switch, while "tail switch" measn the estimation on the egress switch. Comparing the first two scenarios, the form (a) shows that the iperf measurement nearly achieves ideal throughput (i.e., 1 Mbps), since the bandwidth of each link is set as 1 Mbps. The linear-3 has slightly better performance, which probably results from less end-to-end delay. From form (b) of the first two cases, both header and tail switch has estimation over 1 Mbps, with tail slightly closer to the iperf result. Also, the iperf nealy has no variation among the 10 trials, and tail estimation has less variation than the head version. This indicates that using the switch closer to the destination host can achieve better results.

On the other hand, the scenario (Figure 10) demonstrates fluctuated throughput on form (a), meaning the contention affects the throughput heavily. From the form (b) we can acquire the same conclusion by looking at the variation of the three throughput values.

To further illustrate the accuracy of estimation, we use another figure, Figure 11, to show the results in the contention scenario. For each of the 10 repeatitions, we plot the iperf, header, and tail throughput values. It's clear that the estimation value is close enough to the iperf measurement, with tail results slightly better than head estimation.

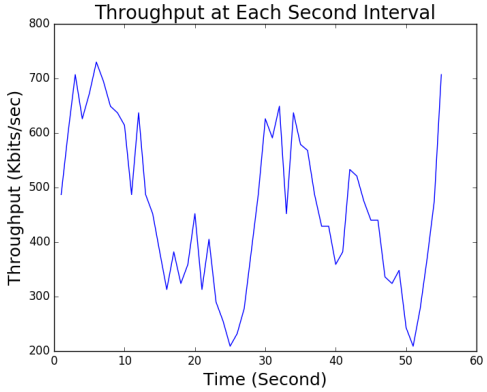


(a) Iperf Result Timeline

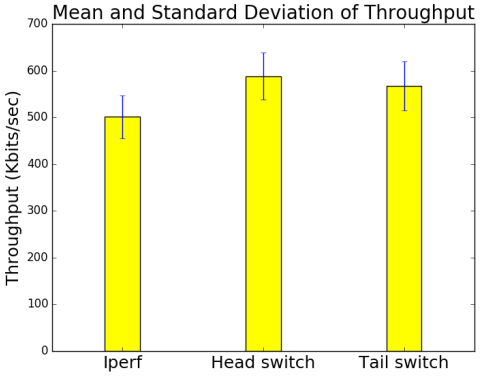


(b) Mean Throughput

Figure 9: Result of Linear-100 Topology



(a) Iperf Result Timeline



(b) Mean Throughput

Figure 10: Result of Linear-100 with Contention

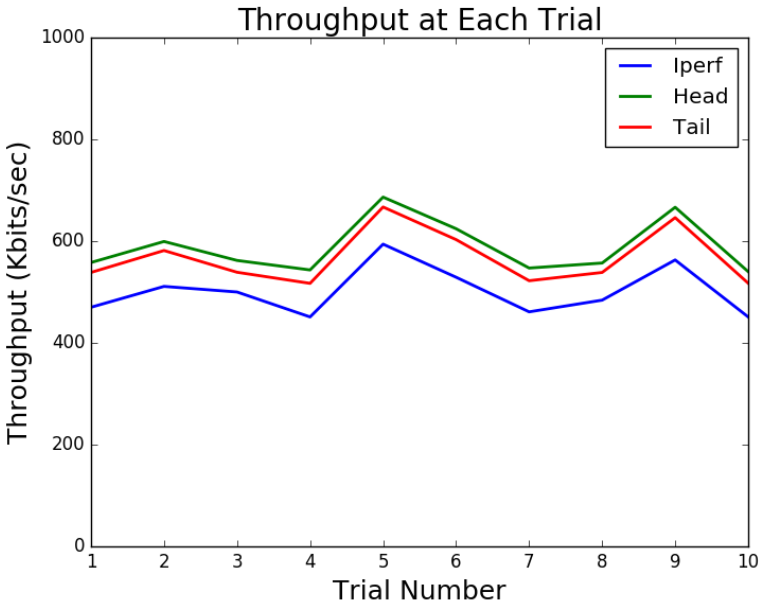


Figure 11: Linear-100 with Contention Each Trial

5 Conclusion and Future Work

From the simple experiments in this work, we can conclude that the statistics information on the switch table entires can be used to estimate the flow's throughput, especially those close to the sink host (i.e., edge switches). However, in a large-scale and complicated network topology, choosing which switch to monitor is not a trival task. This is because the flows in the network has different source and destination hosts, and often share the same links. Also, the large number of switches makes it impossible to monitor every edge switch. Thus, given a network topology and flows, how to decide which switches to pull the stats to achieve optimal overall estimation is an interesting future work that requires further research.

References and Notes

1. K. Kirkpatrick, *Commun. ACM* **56**, 16 (2013).

2. N. McKeown, *et al.*, *SIGCOMM Comput. Commun. Rev.* **38**, 69 (2008).

3. D. J. Hamad, K. G. Yalda, I. T. Okumus, *Information Technology and Systems* pp. 7–11 (2015).

4. N. L. Van Adrichem, C. Doerr, F. A. Kuipers, *Network Operations and Management Symposium (NOMS), 2014 IEEE* (IEEE, 2014), pp. 1–8.

5. N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, N. McKeown, *Proceedings of the 8th international conference on Emerging networking experiments and technologies* (ACM, 2012), pp. 253–264.

6. C.-H. Hsu, U. Kremer, *Workshop on Profile and Feedback-Directed Compilation (PFDC), Paris, France* (Citeseer, 1998).