Estimation of Raw Packets in SDN

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 SDN based networks, for network management such as monitoring, performance tuning, enforcing security, configurations, calculating QoS metrics etc. a certain fraction of traffic is responsible. It consists of packets for many network protocols such as DHCP, MLD, MDNS, NDP etc. Most of the time these packets are created and absorbed at midway switches. We refer to these as raw packets. Cumulative statistics of sent and received traffic is sent to the controller by OpenFlow compliant switches that includes these raw packets. Although, not part of the data traffic these packets get counted and leads to noise in the measured statistics and thus, hamper the accuracy of methods that depend on these statistics such as calculation of QoS metrics.

In this paper, we propose a method to estimate the fraction of the network traffic that consists of raw packets in Software Defined Networks. The number of raw packets transferred depends on the number of switches and hosts in the network and it is a periodic function of time. Through experiments on several network topologies, we have estimated a way to find a cap on the generated raw packets in the network, using spanning tree information about the topology.

Keywords: Raw packets · OpenFlow
Software Defined Networks (SDN)

1 Introduction

The traditional networks with their closed and proprietary network devices by their rigidity have narrowed the possibilities of any innovation and improvement. Further, the strong coupling between data and control plane of network devices, especially switches and routers has restricted the development of new functionalities in the existing networks.

To change the current affairs, Software Defined Networks (SDN), an emerging paradigm in networking advocates rifting of control and data plane, for promoting network programmability by splitting network's control logic from the underlying network devices. The state and statistics of the network greatly affect the decision making process while the controller exerts its centralized control.

OpenFlow 1.3 [1,2] defines structure and semantics for multipart request and response messages which are used by the controller to query statistics from an OpenFlow compliant switch. The counters of received and transmitted packets for flows as well as ports statistics include count of packets which are responsible for network management such as network monitoring, traffic measurement, pushing configurations etc. We refer to these packets as raw packets. Some of the protocols which generate raw packets are NDP, DHCP etc. Often these packets are generated and absorbed at the intermediate switches and are not part of the end-to-end data traffic. Therefore, accuracy of many of the functionalities of the controller that depend on pure end-to-end packet statistics such as calculation of QoS metrics (delay, packet loss, bandwidth etc.), bandwidth management, congestion avoidance, detection and mitigation etc. is hampered. Thus, an estimate of the fraction of network traffic that consists of raw packets is needed.

In this work, we present the intuitions that can help us to estimate fraction raw packet traffic in the network. We also explain the experiments that we conducted to validate those intuitions. Further, we present an implementation and evaluation of a prototype using Ryu controller [3] running on the top of Open vSwitches [4] emulated using Mininet [5,6].

2 Raw Packets

In this section, we define raw packets and discuss its impact in SDN and the extent to which it affects statistics.

2.1 Definition

We define raw packets as fraction of traffic in the network that is responsible for management of network which includes as network monitoring, performance management, enforcing security policies, traffic measurement, pushing configurations etc., but not a part of the end-to-end data traffic. Simply put, it is control, non-user generated traffic.

For example, the switches and routers perform network discovery, multicast listeners discovery etc. and hosts request networking parameters from DHCP servers etc. Many of these packets generated by protocols like NDP, CDP etc. are generated and absorbed at switches while packets generated by protocols like MDNS, DHCP are absorbed at servers. Hence, they are not part of the end-to-end data traffic.

2.2 Impact of Raw Packets in Statistics

Globally, a lot of network traffic comprises of raw traffic [7]. In our initial explorations, we found out that around 3200–3500 packets are generated every 30 min even in a small emulated network of 2 hosts and 2 switches, which is around 3–9% of the total traffic. Cumulatively, over a period of time, these statistics affect the calculation of metrics such as queue size, bandwidth, packet loss etc.

in a Software Defined Network. For example, OpenNetMon [8] estimates per-flow packet loss by polling each switchs port statistics. But because switches send cumulative statistics (that includes raw packets), the reported statistics is unable to differentiate between raw packets and end-to-end user data traffic.

Further, each of the protocols have separate rate of raw packet generation. For example, MDNS queries are generated quite frequently as compared to ICMP Router Solicitation packets. So, instead of identifying a list of triggers of raw packet generation for so many protocols, (which will be even more in a realistic network), an experimental method to detect a periodic time interval is more practical and deployable (Table 1 and Fig. 1).

Table 1. Packet generated by different protocols

Protocols	Packets
ARP	70
DHCP	528
ICMP	928
IGMP	24
MDNS	1578
Total	3583

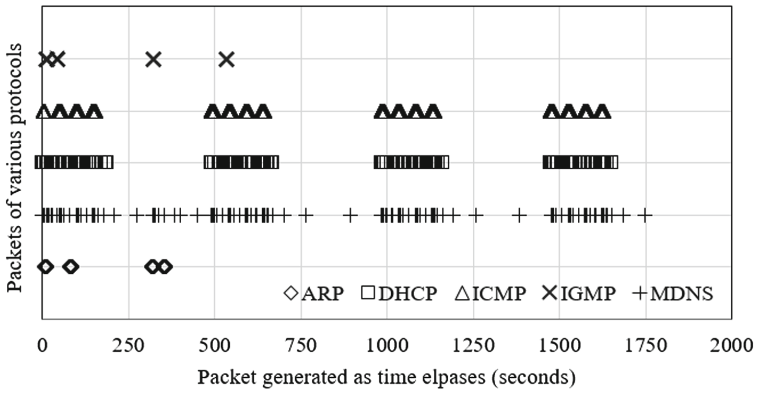

Fig. 1. Rate of raw packet generation of different protocols

2.3 Impact of Raw Packets in SDN

Several background services in SDN controller are responsible for generating a lot of raw packets for several operations like topology discovery, monitoring via packet injection, configuration pushing etc. Therefore, as compared to traditional networks SDN based networks generate more raw packets. Pakzad et al. [9] have shown that even for medium sized topologies such as a tree topology, (d = 4, f = 4, switches = 85 and ports = 424), there can be as many as 424 LLDP packets generated for discovering links part of which are generated and injected by the controller. At first sight, one may argue that the controller already knows about these packets and can separate these counters from received statistics, but many of these packets are generated at switches also such as LLDP Packet-Out message for each port on each switch in OFDP [10] and thus, the controller is unable to estimate the links where these packets are generated.

Because of criticality of load on controller and performance for the scalability of a Software Defined Network [11], authors in [12] have advocated a need to make a trade- off between resource overhead and measurement accuracy. Thus, PayLess [13] proposes a frequency adaptive statistics collection scheduling algorithm and Pakzad et al. propose a new approach to reduce processing cost due to topology discovery in the controller with a minimum reduction of 67% in terms of messages. In their recent work [14], Pakzad et al. have conducted a range of experiments on the OFELIA SDN testbed [15], on a network topology across Italy, Spain, Belgium and Switzerland, the results of which highlight that considerable amount of LLDP packets are generated. For auto configuration in SDN [16], extensions to current protocols such as DHCP-SDN have been proposed that will lead to even greater fraction of raw traffic. These works emphasize that considerable amount of raw traffic is generated that distorts traffic statistics to a greater extent.

3 Intuition

Here, we present the intuitions that can help us to estimate fraction raw packet traffic in the network. We also explain the experiments that we conducted to validate those intuitions.

3.1 Hypothesis

Periodic message exchanges. The message exchanges for network management are periodic in nature. For example, every 15 s a router may send messages to its adjacent routers for network discovery. Therefore, with the help of the time period of the periodic function and number of cycles elapsed, one can estimate roughly the number of messages exchanged.

Raw packets generated proportional to network devices. The number of raw packets transferred in a subnet should be directly proportional to the number of switches and hosts in the network. Thus, the total number of raw packets in the network can be estimated.

Packet flow via spanning tree. Number of packets through each link in the network can be estimated using the spanning tree information about the topology.

3.2 Validation

Experimental Setup. The network is emulated using a Network Emulator, Mininet which emulates any number of virtual end-hosts, routers, switches, and links on a 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) Open vSwitch Kernel switches and Ryu controller to handle their control plane.

We emulate the network topology using L2 learning switches. These switches memorize source-port mapping by examining each packet. By this mechanism each MAC address gets bound with a port. Afterwards, if the destination address of a new packet is found to be associated with some port then, the packet is pushed to the given port, otherwise it is flooded on all the ports of the switch. No other traffic, except raw traffic is generated in such a network.

Periodic message exchanges. We emulate a small network to detect if messages exchanged for network management are periodic in nature (Fig. 2).

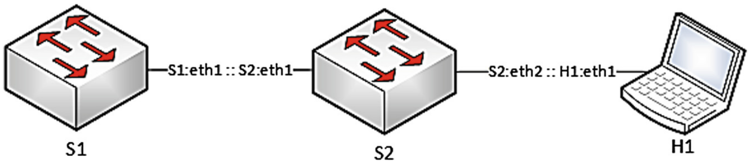

Fig. 2. A network with a host and two switches

The packets exchanged between two switches and between a switch and a host are monitored by sending PortStats query from the controller every 5 s. As shown in Fig. 3, we notice that the number of packets exchanged are periodic in nature. Thus for an experimentally estimated time period, the number of raw packets exchanged between a pair of network devices remains almost constant (doesnt depend on the count of switches and hosts in the network).

Raw packets generated proportional to network devices. As we increase the number of switches and hosts in the network, we see a linear increase in the number of messages exchanged (Fig. 4).

Packet flow via spanning tree. For different topologies such as star, tree etc. while analyzing the PortStats of the switches we find that the number of raw packets exchanged for each port of a switch is proportional to number of switches and hosts connected to the switch through that port in the spanning (Fig. 5).

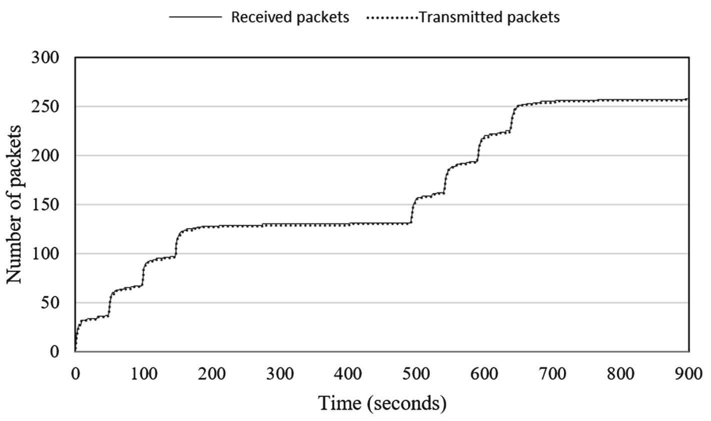

Fig. 3. Exchanged packets between one host and one switch

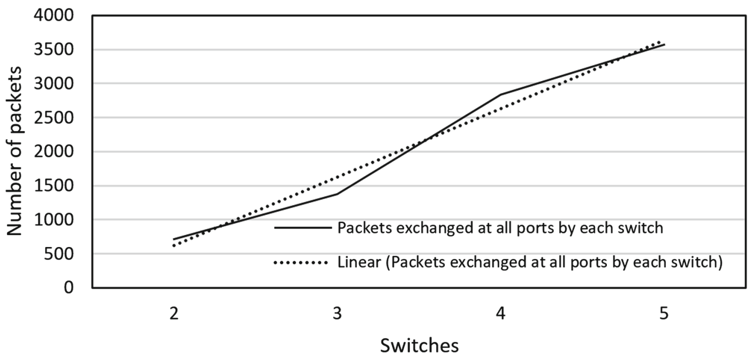

Fig. 4. Total packets exchanged at all ports by each switch as a function of number of hosts in the network

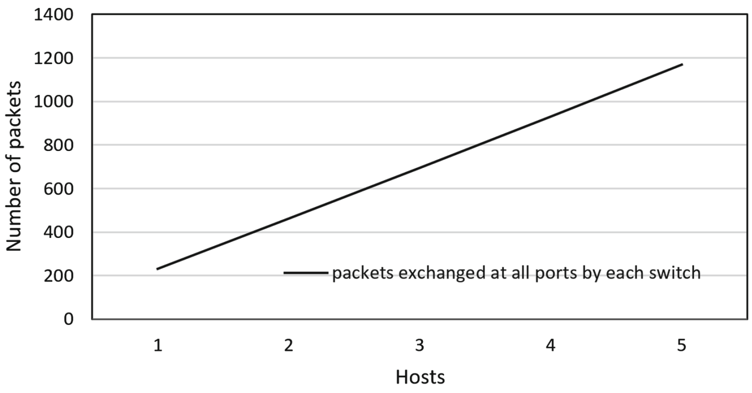

Fig. 5. Packets reported at a port as a function of number of switches connected in the spanning tree

4 Proposed Methodology

In this section, we present the steps of our proposed methodology, the way to get spanning tree information using the controller and the way to estimate constants of the formula.

4.1 Steps

It consists of following steps:

1. Request spanning tree information T of the entire network from the controller.
2. By iterating over each interface of every network device, calculated the number of network devices attached to it.
3. Using the statistics of exchanged packets, timer period (τ) and A_τ and B_τ are estimated. A_τ denotes the number of raw packets exchanged between two switches. And B_τ denotes the number of packets exchanged between a host and a switch in τ time.
4. Then, for a given time t, the total count of the exchanged packets between network devices is calculated using the following formula:

$$N = (A_\tau \times \alpha + B_\tau \times \beta) \times (t/\tau)$$

α: # switches in interface's subnetwork
β: # hosts interface's subnetwork
A_τ: # raw packets exchanged between two switches in τ time,
B_τ: # raw packets exchanged between a host and a switch in τ time

Algorithm 1. Raw packet Estimation

```
1: procedure RAWPACKETS($T, \alpha, \beta, A_\tau, B_\tau, t, \tau$)
2:     for ∀ switches $\alpha_i$ in spanning tree $T$ do
3:         for ∀ switch ports $p_j$ do
4:             $N(p_j) = (A_\tau \times T\alpha_i p_j n + B_\tau \times T\beta_i p_j n) \times (t/\tau)$
5:         end for
6:     end for
7: end procedure
```

In the above algorithm, $T\alpha_i p_j n$ is the number of switches connected to p_j of α_i and $T\beta_i p_j n$ is the number of hosts connected to p_j of α_i.

4.2 Getting Spanning Tree Information

We run spanning tree protocol at the Ryu controller using OpenFlow 1.3 [2]. By sending a Port Modification message to the switch, it is possible to control the following (Table 2):

Table 2. Port settings allowed in OF 1.3 used for STP implementation

Values	Description
OFPPC PORT DOWN	Status, disabled by service personnel
OFPPC NO RECV	Rejects all packets received
OFPPC NO FWD	No forwarding from the port
OFPPC NO PACKET IN	Packet-In messages, not discharged in case of table-miss

Initially, to receive BPDU packets at the controller, we install flow entry that sends Packet-In of BPDU packets in each switch. To control sending/receiving of BPDU packets, MAC learning is employed. For various STP states the following settings are set up (Table 3):

Table 3. STP state configurations

Status	Port configuration	Flow entry
DISABLE	NO RECV/NO FWD	No setting
BLOCK	NO FWD BPDU	Packet-In, drop packets other than BPDU
LISTEN	No setting	BPDU Packet-In, drop packets other than BPDU
LEARN	No setting	BPDU Packet-In, drop packets other than BPDU
FORWARD	No setting	BPDU Packet-In

When connection between each OpenFlow switch and the controller is completed, exchange of BPDU packets starts and root bridge selection, port role setting, and port state change takes place.

4.3 Estimating Time Period and Other Constants

Time period, τ was estimated by emulating a traffic free network for a long duration of time and analyzing the statistics of packets exchanged. Every switch was polled every 5 s for PortStats for 30 min. The time period of the periodic pattern observed is taken as τ. We take A_τ as average number of packets that are exchanged between two switches and B_τ as the average number packets exchanged between a host and a switch in that time period. If we want to know the number of raw packets exchanged at a time which is not a multiple of v, we can have those values of A_τ and B_τ looked up, e.g. from Fig. 3 at that point of time.

5 Evaluation and Inferences

Here we present the results of the application of above methodologies and the error rate with which we were able to report the raw packets accurately.

5.1 Estimation of Constants

As shown in Fig. 3, we noticed that on an average 345 packets are exchanged between two switches in every 450 s. We take this value as A_τ. Even for a network of varied size and complexity, we found this value to be, in range of 340 to 350 packets, almost constant. Therefore the time period of the network (τ) is 450 s.

Similarly, for calculating the number of packets exchanged between a switch and a host we noticed that on an average 115 packets (τ) are exchanged between a switch and a host in every 450 s.

5.2 Setup I

In the setup we emulate the topology as shown in Fig. 6. For estimating raw packets at port 1 (right side) of switch S1, we see that there are two switches and three hosts connected to the switch via port 1. The time period as calculated above is 15 min. Therefore, the estimated number of raw packets for 30 min is $(345 \times 2 + 115 \times 3)(900/450) = 2070$, which is within 3% experimental error rate. Values for other ports are calculated in a similar way as tabulated in Table 4.

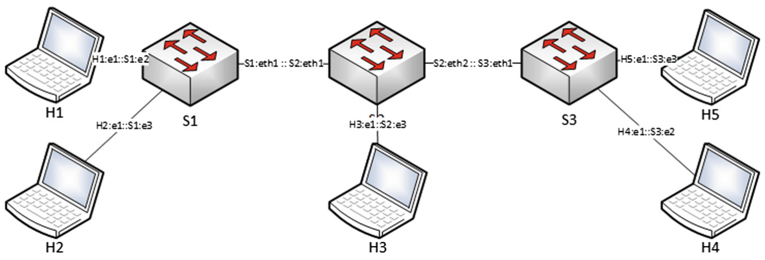

Fig. 6. Experimental Setup 1

Table 4. Raw packet estimation for Setup 1

Switch/port	Switch-1	Switch-2	Switch-3
Port-1	2007	1130	2052
Port-1 estimation	2070	1150	2070
Port-1 error	3.14%	1.77%	0.88%
Port-2	224	1130	223
Port-2 estimation	230	1150	230
Port-2 error	2.67%	1.77%	3.13%
Port-3 225	224	224	
Port-3 estimation	230	230	230
Port-3 error	2.22%	2.67%	2.67%

5.3 Setup II

Here we emulate a tree topology as shown for 900 s. Therefore, number of cycles elapsed is 2. For port 2 of switch S1, we have one switch and three hosts, therefore the number of raw packets exchanged is $(345 \times 1 + 230 \times 3)(900/450) = 2070$, which is within 2.57% experimental error rate (Table 5 and Fig. 7).

Table 5. Raw packet estimation for Setup 1

Switch/Port	Switch-1	Switch-2	Switch-3	Switch-4
Port-1	2851	471	477	474
Port-1 estimation	2070	460	460	460
Port-1 error	3.19%	2.34%	3.56%	2.95%
Port-2	2833	470	473	480
Port-2 estimation	2070	460	460	460
Port-2 error	2.57%	2.12%	2.74%	4.16%
Port-3	2826	470	468	476
Port-3 estimation	2070	460	460	460
Port-3 error	2.34%	2.12%	1.71%	3.36%

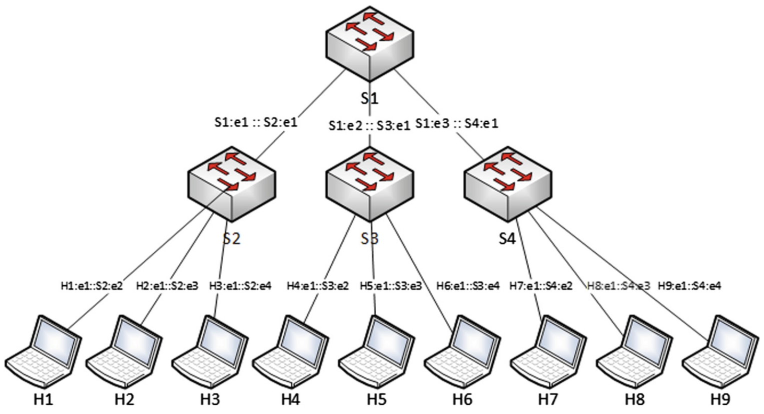

Fig. 7. Experimental Setup 2

6 Conclusion

We have proposed a method to estimate the fraction of raw packets in a given SDN network, which is around 3–9% of the total traffic. The estimation technique proposed in this paper will keep controller informed about the flow of raw packets in the network. This information can be utilized to increase the accuracy of the various techniques like OpenNetMon [8], OpenTM [17] etc. which rely on the cumulative statistics of packet transmitted and received from the switches. The method can be even further generalized for other network devices such as routers as well.

References

1. Mckeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J., Louis, S.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69 (2008)
2. Pfaff, B., Lantz, B., Heller, B., Barker, C., Cohn, D., Talayco, D., Erickson, D., Crabbe, E., Gibb, G., Appenzeller, G., Tourrilhes, J., Pettit, J., Yap, K., Poutievski, L., Casado, M., Takahashi, M., Kobayashi, M., McKeown, N., Balland, P., Ramanathan, R., Price, R., Sherwood, R., Das, S., Yabe, T., Yiakoumis, Y., Kis, Z.L.: OpenFlow Switch Specification 1.3 (2012). https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf
3. Ryu SDN Framework. https://osrg.github.io/ryu/
4. Open vSwitch. http://openvswitch.org/
5. Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet. http://mininet.org/
6. Lantz, B., Heller, B., McKeown, N.: A network in a laptop. In: Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets 2010, p. 16 (2010)
7. Official Google Blog: Google Public DNS: 70 billion requests a day and counting. https://googleblog.blogspot.in/2012/02/google-public-dns-70-billion-requests.html
8. Van Adrichem, N.L.M., Doerr, C., Kuipers, F.A.: OpenNetMon: network monitoring in OpenFlow software-defined networks. In: IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium, Management in a Software-defined World (2014)
9. Pakzad, F., Portmann, M., Tan, W.L., Indulska, J.: Efficient topology discovery in software defined networks. In: 2014 8th International Conference on Signal Processing and Communication Systems, ICSPCS 2014, Proceedings, May 2016 (2014)
10. OpenFlowDiscoveryProtocol GENI: geni. http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol
11. Tootoonchian, A., Gorbunov, S., Ganjali, Y., Casado, M., Sherwood, R.: On Controller Performance in Software-Defined Networks
12. Moshref, M., Yu, M., Govindan, R.: Resource/Accuracy Tradeoffs in Software-Defined Measurement
13. Chowdhury, S.R., Bari, M.F., Ahmed, R., Boutaba, R.: PayLess: a low cost network monitoring framework for software defined networks. In: 2014 IEEE Network Operations and Management Symposium, pp. 1–9 (2014)
14. Pakzad, F., Portmann, M., Tan, W.L., Indulska, J.: Efficient topology discovery in OpenFlow-based software defined networks. Comput. Commun. **77**, 52–61 (2016)
15. Su, M., Bergesio, L., Woesner, H., Rothe, T., Kpsel, A., Colle, D., Puype, B., Simeonidou, D., Nejabati, R., Channegowda, M., Kind, M., Dietz, T., Autenrieth, A., Kotronis, V., Salvadori, E., Salsano, S., Krner, M., Sharma, S.: Design and implementation of the OFELIA FP7 facility: the European OpenFlow testbed. Comput. Netw. **61**, 132–150 (2014)

16. Katiyar, R., Pawar, P., Gupta, A., Kataoka, K.: Auto-configuration of SDN switches in SDN/non-SDN hybrid network. In: Proceedings of the Asian Internet Engineering Conference, pp. 48–53 (2015)
17. 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