# OpenTM: Traffic Matrix Estimator for OpenFlow Networks

Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali

Department of Computer Science
University of Toronto, Toronto, ON, Canada
{amin,monia,yganjali}@cs.toronto.edu

**Abstract.** In this paper we present OpenTM, a traffic matrix estimation system for OpenFlow networks. OpenTM uses built-in features provided in OpenFlow switches to directly and accurately measure the traffic matrix with a low overhead. Additionally, OpenTM 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. We explore several algorithms for choosing which switches to query, and demonstrate that there is a trade-off between accuracy of measurements, and the worst case maximum load on individual switches, *i.e.*, the perfect load balancing scheme sometimes results in the worst estimate, and the best estimation can lead to worst case load distribution among switches. We show that a non-uniform distribution querying strategy that tends to query switches closer to the destination with a higher probability has a better performance compared to the uniform schemes. Our test-bed experiments show that for a stationary traffic matrix OpenTM normally converges within ten queries which is considerably faster than existing traffic matrix estimation techniques for traditional IP networks.

## 1 Introduction

A traffic matrix (TM) represents the volume of traffic between origin-destination (OD) pairs in a network. Estimating the point-to-point TM in a network is an essential part of many network design and operation tasks such as capacity planning, routing protocol configuration, network provisioning, load balancing, and anomaly detection.

Direct and precise measurement of TM in large IP networks is extremely difficult, if not infeasible, due to the large number of OD pairs, the high volume of traffic at each link, and the lack of measurement infrastructure [1]. Previous works infer the TM (a) indirectly from link loads [2,3], (b) directly from sampled flow statistics (*e.g.*, using Cisco NetFlow) [4, 5], or (c) using a combination of both [1]. Indirect methods are sensitive to the statistical assumptions made in their models and are shown to have large errors [6]. Direct methods can be quite attractive due to their high accuracy levels. However, the lack of required measurement infrastructure and the prohibitively large overhead imposed on the network components are two main drawbacks of direct measurements.

A. Krishnamurthy and B. Plattner (Eds.): PAM 2010, LNCS 6032, pp. 201–210, 2010.

In this paper, we revisit the TM estimation problem using direct measurements in the context of OpenFlow-based networks [7]. OpenFlow is an open standard that makes any changes to the network control plane very easy by separating the data and control planes. An OpenFlow network consists of OpenFlow switches (data plane) managed by a logically centralized OpenFlow controller (control plane) which has a network-wide view. OpenFlow's unique features remove the prohibitive cost of direct measurements for TM estimation. Unlike commodity switches, flow level operations are streamlined into OpenFlow switches which lets us query for flow statistics, enabling access to accurate flow statistics.

Taking advantage of these features, we have designed OpenTM, a TM estimator for OpenFlow networks. OpenTM reads byte and packet counters kept by OpenFlow switches for active flows and therefore incurs a minimal overhead on network elements. At the same time the highest level of accuracy is preserved, because the TM is derived directly without making any simplifying mathematical and statistical assumptions. Our work shows the possibility of direct measurement of TM with the least overhead as long as the infrastructure (OpenFlow here) provides the appropriate feature set for measurements. We note that the scope of our work is limited to the networks where OpenFlow can be deployed, *i.e.*, where maintaining per-flow counters is likely tractable.

For different flows, OpenTM can query any switch along the flow path. This choice, however, can affect the accuracy of the measurement as well as the load on individual switches. We present several strategies for choosing which switch to query at any point of time. Even though all these schemes result in TM estimations with very small errors, our analysis and experiments show that there is a trade-off between the accuracy of the TM measurements and the maximum query load on individual switches, *i.e.*, the perfect query distribution sometimes results in the worst estimate, and the best estimation can lead to the worst query distribution amongst switches.

We have implemented OpenTM as an application for NOX [8], an open-source OpenFlow controller. We study OpenTM's performance on a small testbed of OpenFlow switches. Even though using a small testbed for evaluation has its own shortcomings, we believe that most results would not be significantly different in larger networks. Our results show that in a system with a stationary traffic matrix, OpenTM normally converges within 10 queries to a value within 3% of the average rate which is notably faster than existing techniques for traditional IP networks.

The contributions of this work are two-fold. First, we present the design and implementation of OpenTM for OpenFlow-based networks. Based on the evaluation, we argue that low-overhead accurate TM estimation is feasible using direct measurements in a setting where the switches keep track of flow statistics. Second, we explore the idea of constructing the TMs from switch-level measurements, where the choice of which switch to query can be decided at runtime. To the best of our knowledge, this is in contrast to the existing techniques that usually instrument all ingress/egress links leading to an very uneven measurement

load on the boundary switches or routers (switches internal to the network have a very little measurement load).

## 2 Design

Direct measurements in large traditional IP networks is prohibitively costly due to the processing required to handle the large volume of traffic at each interface [1]. On the other hand, OpenFlow switches keep track of active flows in the network and update per flow counters. The measurement infrastructure that OpenFlow provides enables direct and precise flow measurements without packet sampling or incurring any prohibitive overhead on switches. We take advantage of these features to present OpenTM's design in this section.

OpenTM's logic is quite simple. It keeps track of all the active flows in the network, gets the routing information from the OpenFlow controller's routing application, discovers flow paths, and periodically polls flow byte and packet-count counters from switches on the flow path. Using the routing information, OpenTM constructs the TM by adding up statistics for flows originated from the same source and destined to the same destination[1]. Using the information available to an OpenFlow controller, OpenTM can create different types of TMs with different aggregation levels for sources and destinations. Our implementation of OpenTM computes the TM for switches, but the implementation can be easily augmented to derive other TM types described in [9].

The total number of queries generated by OpenTM during each querying interval is bounded by the number of active flows in the network. It is commonly believed that the number of concurrently active flows in large enterprise IP networks is small. According to the data from the 8000-host network at LBNL, the total number of active flows in their network never exceeds 1200 in any second [10]. The data from the Stanford Computer Science and Electrical Engineering network with 5500 active hosts shows that their number of active flows stays well below 10000 [11]. Currently, our system generates a single query for a single source-destination IP pair. As an improvement, a single query can be generated for all flows sharing the same path, as long as the IP addresses could be aggregated.

Different switches on the path may observe different rates for a given flow due to packet loss. We consider the last switch on the flow path to be the point of reference since this is what is seen by the receiver. Consequently, we query the last switch on the path for the most accurate TM. However, this strategy imposes an uneven and substantially high amounts of load on the first/last switches and does not scale well. We expect to get close statistics if other switches on the flow path are queried since packet loss is negligible in enterprise networks (where OpenFlow is designed for). Based on this observation, we propose different switch querying strategies: *(a)* querying the last switch, *(b)* querying switches on the

[1] Multipath routing, routing changes or hot potato routing do not affect the correctness of OpenTM, because OpenTM coordinates with the controller routing application to discover any change in flow paths.

flow path uniformly at random, *(c)* round-robin querying, *(d)* non-uniform random querying that tends to query switches closer to the destination with a higher probability, and *(e)* querying the least loaded switch.

Querying the last switch results in the most accurate TM, but imposes a substantial load on edge switches. Uniform random querying of switching elements of a given flow's path evenly distributes the load amongst switches as long as all switches are equally capable. The price, however, is losing some accuracy. Round-robin querying deterministically queries switches on a round-robin fashion. On average, we expect both uniform random querying and round-robin querying to behave similarly, but round-robin querying may result in synchronization in querying, because the same switch might be queried by several flows simultaneously. Using a non-uniform distribution for querying switches gives us control over the accuracy and the load of OpenTM. A distribution which chooses last switches in the path with a higher probability, results in a more accurate TM but imposes more load on those switches. In our experiments, for non-uniform querying, we randomly select two switch along the flow path and query the one closer to the destination. Querying the least loaded switch evenly distributes queries among all switches in the network, contrary to the uniform random querying method which only distributes queries among switches on individual flow paths. In Section 5, we compare these methods with each other.

The frequency at which OpenTM queries switches for statistics is another factor that directly affects the accuracy and overhead of TM estimation. Querying more frequently results in a more accurate TM but with the cost of added overhead. Here we only consider fixed length intervals for querying different switches for all the flows. Switch querying interval can be adaptively adjusted for each source-destination IP pair based on the flow and network dynamics (*e.g.*, round trip time, available bandwidth). The relation between an efficient querying frequency and flow and network dynamics is outside the scope of this work.

## 3 Implementation

We implemented OpenTM as a C++ application for NOX [8], an open-source OpenFlow controller designed to simplify developing network applications. A NOX application can get notified of all network events (*e.g.*, flow initiation and termination), has access to the routing information, and can interact with the switches in the network. NOX also lets applications interact with each other[2].

In each querying interval, OpenTM queries the network for the statistics of all active IP pairs. Element $(i, j)$ in the TM is then computed by summing up the flow rates that are originated from switch $i$ and are destined to switch $j$. We note that the flow statistic queries do not hit switches at the same time, because flow initiation among OD-pairs are not synchronized.

OpenTM starts querying for statistics periodically once it sees the first flow between an OD-pair and stops querying once all the flows between an OD-pair are expired. To keep track of the number of active IP pairs in the network,

[2] For instance, OpenTM exposes the real-time traffic matrix to other applications.

OpenTM counts the number of TCP/UDP flows between IP pairs. OpenTM increments the mentioned counter upon receiving a `Flow_in` event and decrements it upon receiving the corresponding `Flow_expired` event[3].

Once the IP pair's flow count becomes one, OpenTM fetches the flow path (a list of switches) from the routing application and sends an aggregate statistics query to a switch in the flow path according the desired querying strategy. OpenTM updates the TM when it gets the aggregate query statistics reply back from the network. At this time, if the IP pair flow count is non-zero, OpenTM registers a callback function to query the network for the flow statistics again after a certain period of time[4]. An implicit assumption here is that all packets flowing from the same source to the same destination take the same path. This enables us to query the same set of routers to get statistics for all flows between an IP pair.

OpenTM keeps track of switch loads, so it can choose the least loaded one and optimally balance the queries among all of them. We use the number of outstanding queries on each switch as the load metric. When a switch is queried, a counter that keeps track of the number of outstanding requests on each switch is incremented. Upon receiving the reply back, that counter is decremented. This simple method captures the difference in the processing power of switches. More capable switches can handle more requests and should get more queries compared to the less capable ones. In the following section, we present the results of our empirical evaluation based on our implementation.

## 4 Experiments and Results

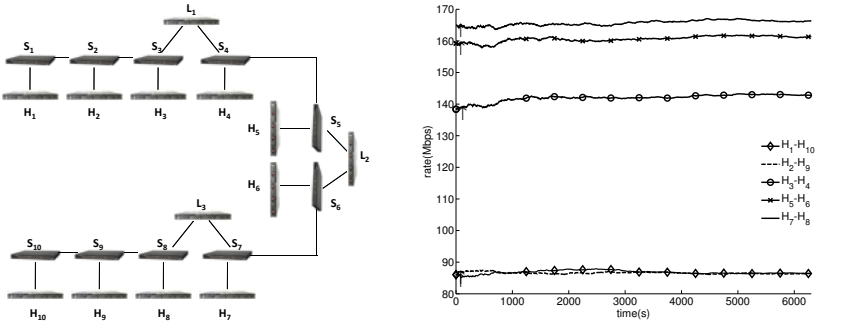
In this section, we present real-time traffic measurements in a testbed to evaluate OpenTM. We study the performance and convergence time of OpenTM. We also compare different switch querying schemes introduced in Section 2.

For our experiments, we use HP DL320 G5p servers equipped with an HP NC326i PCIe dual-port gigabit network card running Debian Lenny and OpenFlow-enabled NEC IP8800/S3640 switches. In all our experiments, we use TCP cubic with the maximum advertised TCP window size set to 20MB. The path MTU is 1500 bytes and the servers send maximum-sized packets. We use the NetEm [12] to emulate network delay and loss, and use Iperf to generate the input traffic.

Our testbed topology is illustrated in Figure 1(a), where $H_i$, $1 \leq i \leq 10$, are host machines, $S_j$, $1 \leq j \leq 10$ are OpenFlow switches, and $L_k$, $1 \leq k \leq 3$ are loss emulator machines. Five OD pairs $H_i$-$H_j$ are created in which host $H_i$ sends TCP traffic to host $H_j$. Specifically, we create 10 TCP flows between each OD pair $H_1$-$H_{10}$, $H_2$-$H_9$, $H_3$-$H_4$, $H_5$-$H_6$, and $H_7$-$H_8$. We add 100ms of delay on the forward path of each flow. The delay emulators are also configured to each

[3] Both `Flow_in` and `Flow_expired` events are fired by the NOX's authenticator application upon flow initiation and termination, respectively. A flow expires when the switch does not see any packets belong to that flow after a specific timeout.

[4] The querying interval which is set to five seconds in our current implementation.




(a) (b)

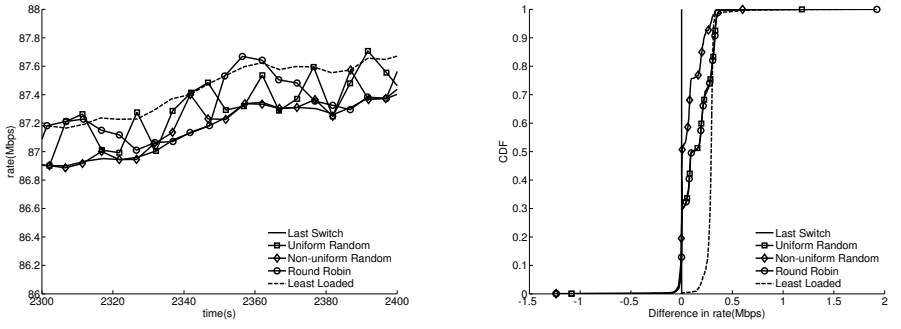
**Fig. 1.** (a) Testbed topology consisting of $H_i$: host machines, $S_i$: OpenFlow switches, and $L_i$: loss emulators. (b) Measured average flow rate for each of the five OD pairs in the network.

emulate 1% packet loss. For the purpose of evaluation, OpenTM logs the flow statistics of all switches every five seconds for a duration of two hours; we then analyze the data offline. Note that this is not the case in the real application where the switches are queried according to the querying scheme chosen by the network operator. Since we are interested in studying the system in equilibrium, we remove the first 15 minutes of our data as the warm up period.

We start with a very basic question: How fast does OpenTM rate measurements converge? For an element $f$ in the TM, let us assume we have $t$ queries. The $i$-th query, $r_i$, is the average rate from the beginning of the measurement up to that point in time. We define the *convergence point* $c$ as the first query for which all the proceeding queries are within 3% of the overall mean rate. We note that since the rate is assumed to be stationary in this system[5], such an average exists. Also, by a simple application of the Central Limit Theorem, we can show that the queries will converge to the real average as we increase the number of queries. Roughly speaking, the convergence point is when the estimated rate becomes and remains very close to the overall average rate that we are trying to estimate.

Figure 1(b) shows the average rate over time for each of the five OD pairs. The measurement is performed at the last switch in each path with a querying interval of only five seconds. We can see that in all cases convergence point, marked by vertical arrows in the graph, is within 50 seconds, or just 10 queries. Note that OD pairs $H_1$-$H_{10}$ and $H_2$-$H_9$ receive the least rate as they are traversing the longest path in the topology through three loss emulators hence experiencing

[5] This assumption does not break the generality of our results. We make the stationarity assumption in order to have a well-defined notion of convergence time. However, as long as the changes in system are slower than our convergence rate, OpenTM gives a close estimate of the TM.




(a) (b)

**Fig. 2.** (a) Comparing the querying strategies methods for traffic between $H_1$ and $H_{10}$ with a querying interval of 5 seconds. (b) CDF of difference between querying strategies and the last switch's traffic.

3% drop rate. On the other hand, $H_3$-$H_4$, $H_5$-$H_6$, and $H_7$-$H_8$ only experience 1% drop rate and thus have higher rates.

In the next set of experiments, we compare different querying strategies proposed in Section 2. Figure 2(a) shows the measured average throughput versus time for traffic between $H_1$ and $H_{10}$ when each of the querying strategies are used and the querying interval is five seconds. To make it more visible, the plot is zoomed in and only shows 100 seconds of the measurement. From the figure, it appears that the least loaded method is almost always reporting a rate higher than the last switch method and having the largest estimation error. This is because in our setup, the least loaded switches are mostly the first switch in a flow's path which is before all three drop emulators and hence this method suffers from the most inaccuracy. This is not necessarily the case in other network topologies and it is dependent on the traffic load over the switches.

To better illustrate the difference between querying strategies, Figure 2(b) shows the CDF of differences between each querying strategy and the last switch's rate for traffic between $H_1$ and $H_{10}$ when each of the querying strategies are used and the querying interval is five seconds. The ideal case is to always measure the last switch's rate and hence having zero difference and it is shown by a vertical line at zero in the graph. As expected from the analysis presented in Section 5, the figure shows that the non-uniform random querying method has the best performance, as it tends to query switches closer to the destination with higher probability. Both the round-robin and uniform random querying methods are performing very close to each other, and worse than the non-uniform querying method. As mentioned above, the least loaded method is performing the worse in this case, since in our setup the first switch on the path is almost always the least loaded switch.

Finally, we note that the overall difference between all these schemes is relatively small. In fact, the maximum difference between the best and worst querying schemes is about 2 Mbps, which is about 2.3% of the actual rate (86 Mbps) between $H_1$ and $H_{10}$. This observation suggests that when we do not need extremely accurate TM, any of these querying schemes can be used. Clearly, in this case the least loaded scheme might be the preferred scheme as it minimizes the maximum load among switches. For higher-accuracy requirements, however, one might want to use schemes which favor the last few switches on the path.

## 5 Analysis

In this section we analytically compare the querying strategies proposed in Section 2 in terms of their accuracy in estimating the flow rates between source and destination. Intuitively, as long as there are no packet drops in the network, all measurements from switches should be the same[6], and thus all querying strategies should be very close to each other; our experiments confirm this. However, when there are packet drops in the system, we expect to see differences in the various querying schemes proposed before.

Let us consider a topology similar to Figure 1(a). We are interested in finding the expected value of rate of a given flow $f$. We denote the link between switches $S_i$ and $S_{i+1}$ by $e_i$ and the measured rate corresponding to $f$ over $e_i$ by $r_i$. If $e_i$ has a drop rate $d$, then the rate measured at $e_{i+1}$ will be $\leq r_i \times (1-d)$. Assuming there are $M$ uniform randomly distributed congestion points in the network, each with a drop rate of $d$, we can find the expected rate for each querying strategy as follows. Note that here for simplicity we assume that all links have equal drop probability of $d$.

**Querying the last switch before the destination.** We define the rate between an OD pair as the rate seen by the destination. Assuming negligible packet drops on the link connecting the last switch to the destination node, querying the last switch must give us the rate as seen by the destination regardless of network conditions. We use this rate as the baseline for comparing with randomized querying techniques presented below.

**Uniform random querying.** We first consider the simple case where there is only one congested link in the network and call the measured rate by this method at a time slot $i$ as $R_r(i)$ and the rate at the last switch by $R_t(i)$. There are two possible cases: (1) if the randomly selected switch is between the congested link and the last switch before the destination, then rate scene by the selected switch is same as the rate at the last switch; $R_r(i) = R_t(i)$ (2) if the selected switch is between the source and the congested link, then rate at the selected switch is higher than the rate at the last hop switch before the destination. In particular, $R_r(i) = \frac{R_t(i)}{1-d}$. Hence, $R_t(i) \leq R_r(i) \leq \frac{R_t(i)}{1-d}$. Assuming that the congested link is placed uniformly random over the path then each of the above cases has an

[6] We ignore the difference caused by the delay between switches.

equal probability of one half. If we take the average rate over $N$ queries, the expected rate $R_r = \sum_{i=1}^{N} R_r(i)/N$, will lie exactly in between the two cases; *i.e.*, $R_r = 0.5 \times (R_t + R_t/(1-d))$.

Similarly, if there are $M$ congestion points in the network then we have $R_t(i) \leq R_r(i) \leq R_t(i)/(1-d)^M$ and if we assume that the congestion points are distributed uniformly over the path, then the probability of $R_r(i) = \frac{R_t(i)}{(1-d)^m}$ is $\frac{1}{M+1}$, where $0 \leq m \leq M$ is the number of congestion points that the flow has traversed before reaching the querying switch. Hence,

$$R_{ur} = \frac{R_t}{M+1} \sum_{m=0}^{M} \frac{1}{(1-d)^m} = \frac{R_t}{M+1} \times \frac{1-(1-d)^{M+1}}{d(1-d)^M} \tag{1}$$

**Non-uniform random querying.** In this method, we generate two random numbers $i$ and $j$, $1 \leq i, j \leq N$, where $N$ is the number of switches in a flow's path and query the switch with ID equal to $\max(i, j)$, assuming the switch with larger ID is the one closer to the destination. With same assumptions as the above and in the case that there are $M$ congestion points in the network we have $(M+1)^2$ cases and if we take the average over $N$ queries for large $N$, the expected average rate will be:

$$R_{nr} = \frac{R_t}{(M+1)^2} \left( 1 + 2 \sum_{m=0}^{M-1} \frac{M-m}{(1-d)^m} + \frac{1}{(1-d)^M} \right) \tag{2}$$

**Round-Robin querying.** The expected value of average rate for the round-robin querying method, $R_{rr}$, is similar to the uniform random method since on average $\frac{1}{M+1}$ of queries will have rate $R_t(i)$, $\frac{1}{M+1}$ of queries will have rate $\frac{R_t(i)}{1-d}$ and so on.

**Least-loaded switch querying.** The performance of least-loaded switch querying highly depends on packet processing power of network switches, as well as how network load is distributed amongst them. If switches have equal processing power and load this scheme will perform very similar to uniform random querying. However, in the worst case, the least loaded switch might be the first switch on the path in which case it will lead to the worst case estimation of the rate.

## 6 Conclusion

This paper presents OpenTM, a traffic matrix estimator for OpenFlow networks. OpenTM derives the TM of an OpenFlow network in real-time with high accuracy using direct measurements without packet sampling. OpenTM evenly distributes the statistic queries among all the switches in the network and thus imposes the least overhead on the network. Our evaluation in a testbed using OpenTM implemented as a NOX application shows that OpenTM derives an accurate TM within 10 switch querying intervals, which is extremely faster than existing TM estimation techniques. Despite the limitations of our evaluation and the need for more comprehensive evaluation, we believe OpenTM can be deployed in OpenFlow networks with a very negligible overhead.

## Acknowledgments

This work was partly supported by Cisco Systems and a grant from NSERC. NEC Corporation kindly provided us with the OpenFlow switches. We would also like to thank Bianca Schroeder and the anonymous reviewers for the their valuable feedback.

## References

1. Zhao, Q., Ge, Z., Wang, J., Xu, J.: Robust traffic matrix estimation with imperfect information: Making use of multiple data sources. SIGMETRICS Performance Evaluation Review 34(1), 133–144 (2006)
2. Vardi, Y.: Network tomography: Estimating source-destination traffic intensities from link data. Journal of the American Statistical Association 91(433), 365–377 (1996)
3. Nucci, A., Diot, C.: Design of IGP link weight changes for estimation of traffic matrices. In: Proceedings of the 2004 Conference on Computer Communications (2004)
4. Feldmann, A., Greenberg, A., Lund, C., Reingold, N., Rexford, J., True, F.: Deriving traffic demands for operational IP networks: Methodology and experience. IEEE/ACM Transactions on Networking 9(3), 265–280 (2001)
5. Papagiannaki, K., Taft, N., Lakhina, A.: A distributed approach to measure IP traffic matrices. In: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (2004)
6. Medina, A., Taft, N., Salamatian, K., Bhattacharyya, S., Diot, C.: Traffic matrix estimation: Existing techniques and new directions. SIGCOMM Computer Communication Review 32(4), 161–174 (2002)
7. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: OpenFlow: Enabling innovation in campus networks. SIGCOMM Computer Communication Review 38(2), 69–74 (2008)
8. Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S.: NOX: Towards an operating system for networks. SIGCOMM Computer Communication Review 38(3), 105–110 (2008)
9. Medina, A., Fraleigh, C., Taft, N., Bhattacharrya, S., Diot, C.: A taxonomy of IP traffic matrices. In: SPIE ITCOM: Scalability and Traffic Control in IP Networks II, Boston (August 2002)
10. Pang, R., Allman, M., Bennett, M., Lee, J., Paxson, V., Tierney, B.: A first look at modern enterprise traffic. In: Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement, Berkeley, CA, USA, p. 2 (2005)
11. Naous, J., Erickson, D., Covington, G.A., Appenzeller, G., McKeown, N.: Implementing an OpenFlow switch on the NetFPGA platform. In: Franklin, M.A., Panda, D.K., Stiliadis, D. (eds.) ANCS, pp. 1–9. ACM, New York (2008)
12. Hemminger, S.: Network emulation with NetEm. In: Linux Conference, Australia (April 2005)