FlowSense: Monitoring Network Utilization with Zero Measurement Cost

Curtis Yu[1], Cristian Lumezanu[2], Yueping Zhang[2], Vishal Singh[2], Guofei Jiang[2], and Harsha V. Madhyastha[1]

[1] University of California, Riverside
[2] NEC Labs America

Abstract

Flow-based programmable networks must continuously monitor performance metrics, such as link utilization, in order to quickly adapt forwarding rules in response to changes in workload. However, existing monitoring solutions either require special instrumentation of the network or impose significant measurement overhead.

In this paper, we propose a push-based approach to performance monitoring in flow-based networks, where we let the network inform us of performance changes, rather than query it ourselves on demand. Our key insight is that control messages sent by switches to the controller carry information that allows us to estimate performance. In OpenFlow networks, PacketIn and FlowRemoved messages—sent by switches to the controller upon the arrival of a new flow or upon the expiration of a flow entry, respectively—enable us to compute the utilization of links between switches. We conduct a) experiments on a real testbed, and b) simulations with real enterprise traces, to show accuracy, and that it can refresh utilization information frequently (*e.g.*, at most every few seconds) given a constant stream of control messages. Since the number of control messages may be limited by the properties of traffic (*e.g.*, long flows trigger sparse FlowRemoved's) or by the choices made by operators (*e.g.*, proactive or wildcard rules eliminate or limit PacketIn's), we discuss how our proposed passive approach can be combined with active approaches with low overhead.

1 Introduction

Enterprises are deploying flow-based programmable networks to support diverse performance- or reliability-based application requirements such as deadline guarantees [8], quick failure recovery [4], or fast and reliable big data delivery [5, 10]. In flow-based networks, a centralized controller locally computes the routes that satisfy a set of requirements and installs them remotely in the forwarding tables of switches. To ensure that traffic flows according to the pre-defined goals and to adapt rules quickly to workload or infrastructure changes, the network must continually monitor the utilization of every link.

Flow-based network utilization monitoring must be not only accurate and responsive in detecting variations, but it must also scale with minimal overhead on the network [3]. Existing monitoring techniques do not satisfy all of these goals simultaneously. Active monitoring techniques (*e.g.* SNMP polling) inject measurement probes and require careful scheduling to scalably monitor the entire network. Passive "capture-and-analyze" tools (*e.g.*, SPAN, netflow, tcpdump) need expensive instrumentation and

infrastructure to gather and process measurements. Recently, several tools take advantage of the functionality provided by software-defined networks (SDNs), which allow the controller to poll switches for utilization-based statistics [11, 6]. Though this eliminates the need for additional instrumentation, control packets used for polling still impose overhead.

In this paper, we propose a new approach for high accuracy utilization monitoring with *zero* measurement cost. Rather than rely on on-demand active polling of switch counters, we infer performance *by passively capturing and analyzing control messages between the switches and the centralized controller*. This is made possible by the physical separation of the control and data planes in SDNs. In particular, we use the control messages that notify the controller of changes in network traffic (*e.g.*, flow arrival, flow expiration). Such changes in traffic may result in changes in performance; by detecting the time and magnitude of these changes, the controller can monitor network utilization locally, without additional instrumentation or overhead.

To explore the feasibility of our control traffic based monitoring, we design FlowSense to measure link utilization (the bandwidth consumed by flows traversing the link) in OpenFlow networks [7]. FlowSense relies on PacketIn and FlowRemoved messages, sent by switches to the controller when a new flow arrives or when a flow entry expires. FlowRemoved messages contain information about the size and duration of flows matched against the entry. To compute utilization over an interval, the controller analyzes all PacketIn and FlowRemoved messages corresponding to the arrival of flows and to the expiration of the flows that were active during the interval.

Relying on control traffic to compute network utilization fails when there is little or no control traffic. This may happen due to the properties of data traffic (*e.g.*, long flows that lead to few flow expiration events) or due to measures taken by network operators (*e.g.*, to limit the amount of control traffic and preserve scalability, they install flow rules proactively that potentially never expire). In this paper, we study the feasibility of our monitoring approach, both in terms of effectiveness (*how accurate is it?*) and compatibility with current networks (*how is it affected by traffic patterns and network deployment scenarios?*).

To summarize, our primary contributions are two-fold. First, we introduce a push-based approach to flow-based network performance monitoring with zero measurement cost, where we let the network inform us of performance changes, rather than query it ourselves. We describe FlowSense, a system to measure link utilization that is simultaneously fast, accurate, and imposes no overhead. Using preliminary experiments on a small OpenFlow deployment, we show that the utilization computed using control plane messages closely resembles that measured on the data plane.

Second, we explore the feasibility of FlowSense in today's networks. We use real world traffic measurements to estimate the impact that the properties of data traffic have on the performance of FlowSense. We find that we can refresh link utilization measurements at most as frequently as every few seconds. Although, to compute utilization at any point in time, FlowSense must wait for all the flows active at that time to finish and trigger FlowRemoved, the wait time is reasonable: we can accurately estimate link utilization in under 10 seconds of delay. Since the network deployment can limit the effectiveness of FlowSense, we discuss combining active and passive techniques.

Ultimately, passively capturing control messages can serve as a building block towards more scalable, accurate, flexible, and general flow-based network monitoring.

2 OpenFlow Overview

In this section, we describe the general operation of an OpenFlow network and review research that uses OpenFlow to monitor network performance.

2.1 Operation

We consider a network of OpenFlow-enabled switches that are connected with a logically centralized controller using a secure, lossless TCP connection. The network operates in the following (simplified) way:

Flow arrival. On the arrival of the first packet of a new flow, the switch looks for a matching rule in the flow table and performs the action associated with the rule (*e.g.*, forward, drop). If there is no matching entry, the switch buffers the packet and notifies the controller that a new flow has arrived by sending a PacketIn control message containing the headers of the packet. The controller responds with a FlowMod message that contains a new rule matching the flow that is to be installed in the switch's flow table. The switch installs the rule and forwards the buffered packet according to it. Subsequent packets in the flow are forwarded without triggering PacketIn's.

Flow completion. Each flow table rule is associated with two timeout values that define when the entry should expire: a hard timeout counted from the time at which the entry was installed, and an soft timeout counted from the time of the last packet which matched the entry. When the flow entry expires, the switch notifies the controller by sending a FlowRemoved control message. The FlowRemoved contains, among others, the duration for which the entry was present in the switch, and the number of packets and number of bytes that matched the entry.

2.2 Monitoring with OpenFlow

The OpenFlow protocol provides functions to query switches for the number of packets or bytes in flows matching against a specific rule or traversing a specific port. Prior work relies on this capability to compute utilization in the network [11, 6]. OpenTM [11] measures network-wide traffic matrix by periodically polling one switch on each flow's path and then combining the measurements. Polling a single switch does not impose significant load on the network but may affect accuracy if the switch is not carefully chosen. Jose *et al.* [6] detect heavy hitters by continually adapting polling rules to focus on the flows that are more likely to have high volume. Both approaches have to carefully schedule measurements to limit the polling overhead and maintain reasonable accuracy. FlowSense, on the other hand, incurs zero measurement cost because it relies on control traffic that the switches already send to the controller.

Ballard *et al.* use OpenFlow to enable flexible monitoring of network traffic for security problems [1]. Their tool, OpenSAFE, directs spanned network traffic towards predefined sinks (*e.g.*, IDS) according to pre-specified policies. While such an approach could be used to compute network utilization (by analyzing the redirected traffic), the overhead it creates by copying all network traffic is prohibitive.

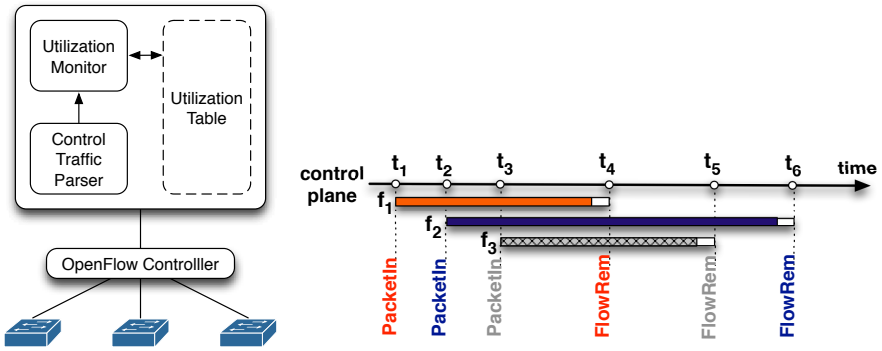

Fig. 1. (left) FlowSense design: Parser module captures control traffic and sends it the monitor. The monitor updates utilization values at every checkpoint according to Algorithm 1. **(right)** Visualization of how link utilization is estimated with the aid of PacketIn and FlowRemoved messages.

3 FlowSense

In this section, we describe the design of FlowSense and how it uses control traffic to measure the utilization of every link in the network.

3.1 Design

FlowSense uses FlowRemoved and PacketIn messages to compute network utilization on inter-switch links. FlowRemoved's are triggered by switches when flow entries expire, and they inform the controller of several properties of the expired entry. Three of these properties are most important to us: (1) the duration of the entry in the flow table, (2) the amount of traffic matched against it, and (3) the input port of traffic that matches the entry (we do not consider wildcard rules for now). This information helps us infer the number of bytes that the flows that matched this entry contributed to the utilization on the link that ends in the specified input port.

Whenever a flow entry expires and triggers a FlowRemoved message, we add a new checkpoint for the corresponding link. We set the timestamp for the checkpoint as the time at which traffic last matched the expired flow entry. If an entry's soft timeout expires, the checkpoint is the FlowRemoved timestamp minus the soft timeout. If the entry's hard timeout expires, we cannot tell how long the flow was actually active for, so we set the checkpoint as the FlowRemoved timestamp and assume it has been active for the entire flow duration.

At every checkpoint, FlowSense can estimate the contribution to the link's utilization made by flows that matched the expired entry as the ratio of the number of bytes matched against the expired entry to the duration of the flows that matched the entry. However, there may be other active flows on the same link that contribute to the total utilization. FlowSense uses information from PacketIn messages, which are triggered when a new flow arrives at a switch, to infer which flows are active at a given time. To compute the utilization contribution of these active flows, we must wait for them to expire and trigger FlowRemoved's. Thus, we incur a delay in estimating the instant

Algorithm 1 Pseudocode of FlowSense's utilization monitor.

```
1: procedure UTILIZATIONMONITOR(Utilization Table UT, Packet p)
2:     Active_List ← set of p.in_port's active flows
3:     if p is a PacketIn packet then
4:         if p's flow ∉ Active_List then
5:             Flow active_flow
6:             active_flow.flow ← p.flow
7:             active_flow.time ← p.time
8:             Add active_flow to Active_List
9:         end if
10:    else if p is a FlowRemoved packet then
11:        flow ← matching flow from A
12:        Remove flow from Active_List
13:        Checkpoint chkpt
14:        chkpt.time ← p.time
15:        if p was from soft timeout then
16:            chkpt.time ← chkpt.time − p.soft_timeout
17:        end if
18:        chkpt.active ← |Active_List|
19:        chkpt.util ← p.byte_count/p.flow_length
20:        for active c in UT do
21:            if c.time is between flow.time and chkpt.time then
22:                c.active ← c.active − 1
23:                c.util ← c.util + chkpt.util
24:            end if
25:            if c.active = 0 then
26:                Declare c final and inactive
27:            end if
28:        end for
29:        Insert chkpt into UT
30:    end if
31: end procedure
```

total utilization on a link at a checkpoint. We evaluate the magnitude of this delay in Section 4.

Figure 1(right) illustrates an example scenario for our estimation of link utilization as above. In this example, f_1, f_2, and f_3 are flows that start at times t_1, t_2, and t_3, and t_4, t_6, t_5 are the times at which those flows end; FlowSense determines the start and end times based on PacketIn and FlowRemoved messages. If f_1, f_2 and f_3 had utilizations of 10, 20 and 40 MBps, then, when the first FlowRemoved message arrives at t_4, FlowSense will know the utilization for f_1 by dividing the byte count from the FlowRemoved message by the duration of the flow, and it also creates a checkpoint at t_4. When the FlowRemoved packet at t_5 arrives, flow f_3 ends and its utilization of 40 MBps is recorded and added to the checkpoint at t_4 leaving it with a current known utilization of 50 MBps (the sum of f_1 and f_3). Finally, at t_6, flow f_2 ends and its utilization is added to the checkpoints at both t_4 and t_5 giving the final checkpoint utilizations recorded to be: 70 MBps at t_4, 60 MBps at t_5, and 40 MBps at t_6.

FlowSense consists of two main modules: the control traffic parser and the utilization monitor. The parser captures control traffic and extracts information from FlowRemoved and PacketIn messages. The utilization monitor maintains a utilization table where it records the current utilization value and a list of active flows at all known checkpoints. The monitor updates the table on every new PacketIn or FlowRemoved data received from the parser. Figure 1(left) shows the design of FlowSense.

The algorithm that FlowSense uses for monitoring utilization on a network works

as follows. When the controller receives a PacketIn message, FlowSense creates a new flow and adds it to a list of active flows ($Active\_List$) associated with the new flow's input port. On a FlowRemoved message, FlowSense removes the corresponding flow from $Active\_List$ and creates a checkpoint ($chkpt$) with a timestamp ($chkpt.time$) equal to the current time minus the soft timeout. It then makes note of the number of currently active flows ($chkpt.active$) and uses the utilization of the flow as the starting utilization of the checkpoint $chkpt.util$. Each previously known active checkpoint (c) for the same input port in the Utilization Table (UT) is then checked to see if its timestamp is between the start and end time of the newly ended flow. If it is, then c's number of active flows and utilization are updated. When a checkpoint's number of active flows hits 0, FlowSense declares that checkpoint final and inactive. Finally, FlowSense inserts $chkpt$ into UT for future lookup purposes. Algorithm 1 describes the steps involved in the utilization monitoring in a more detailed manner.

3.2 Limitations

Using control traffic to compute utilization has two limitations. First, we are able to compute utilization only at discrete points in time. These checkpoints are determined by FlowRemoved arrivals at the controller and by the values of the timeouts associated with the expired entry. In Section 4.2, we show that the average difference between consecutive checkpoints on a link is less than two seconds.

Second, how quickly we are able to estimate the utilization at a checkpoint depends on the type of traffic; long flows that last forever can delay indefinitely the computation of utilization. Our results in Section 4.3 show that, if FlowSense is willing to tradeoff 10% of accuracy, it can measure total utilization at a checkpoint in under 10 seconds. We also discuss ways to improve the estimation delay by combining active and passive measurements.

Finally, FlowSense is limited to reporting the average utilization over a flow entry's duration and cannot capture instant utilization at any point in time. Thus, it works best in environments with many short flows, such as data centers or enterprises [2], where the small duration of a flow and the small difference between consecutive checkpoints make the average utilization a good approximation of the instant utilization.

4 Evaluation

We evaluate FlowSense from three perspectives: (1) how accurate are its utilization estimates?, (2) how often can it refresh its estimate for a link's utilization?, and (3) how quickly can it estimate the utilization at a specific time? To answer these questions, we perform experiments using a small OpenFlow testbed and simulations on a real-world enterprise trace.

4.1 Accuracy

To estimate the accuracy of utilization monitoring, we set up a small testbed comprising two OpenFlow switches, A and B, that are connected to each other. hostA is connected to A, and hostB1 and hostB2 to B. Initially, the rule tables of the two switches are empty. When new flows arrive, we always add rules with no hard timeout and a soft timeout of 1s. We use *iperf* to simultaneously perform two data transfers from hostA to

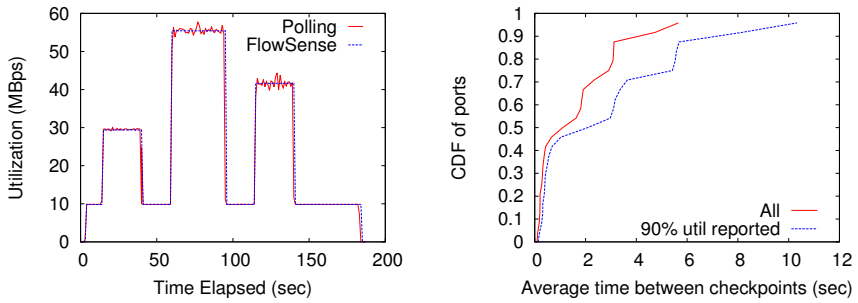

Fig. 2. (left) Accuracy of utilization monitoring. We compare FlowSense's estimates with the values obtained by continually polling the switch counters at 1s intervals; **(right) Granularity of utilization monitoring** for all flows and for flows that have 90% of their utilization reported after 10s. We assume flows are mapped to 24 distinct links.

hostB1 and hostB2 for a period of three minutes. The transfer from hostA to hostB2 has a constant rate of 10MBps, while the transfer from hostA to hostB1 varies across three different rates over time: 20MBps, 45MBps, and 30MBps. Before changing the transfer rate, we briefly interrupt the transfer for a little more than a second to allow the soft timeout to expire and trigger FlowRemoved messages.

We compare the utilization obtained by FlowSense with that gathered from continually polling A and B at 1s intervals. Figure 2(left) presents the results obtained for the link connecting A and B. FlowSense reports utilization values that are similar to those inferred through polling. In comparison to the values obtained with polling, utilization measured with FlowSense shows a small shift to the right because flow entry timeouts have a precision at the granularity of seconds. Thus, it may take up to a second for FlowRemoved to trigger after a timeout expires. Since FlowSense is only working with a single PacketIn and FlowRemoved message per flow, it does not experience the same jittery behavior as the polling method because its readings are an average utilization over that flow's lifetime.

4.2 Granularity

Many applications need to monitor utilization as often as possible to quickly react to traffic changes. How often FlowSense captures utilization depends on the distribution of flows, in particular on how frequently and how rapidly flow entries expire and trigger FlowRemoved's.

To evaluate the granularity of measurements, we simulate FlowSense on a real-world enterprise trace. We use the *EDU1* trace collected by Benson *et al.* [2], capturing all traffic traversing a switch in a campus network for a period of two hours. We identify all network flows (*i.e.*, pairs of IP addresses and application ports) in the trace, along with their start and finish times. The finish time of a flow is an approximation of when the flow entry associated with the flow would expire and trigger a FlowRemoved message in an OpenFlow network. We consider a flow as finished if there is no traffic between the associated endpoints for at least five seconds. We compute the aver-

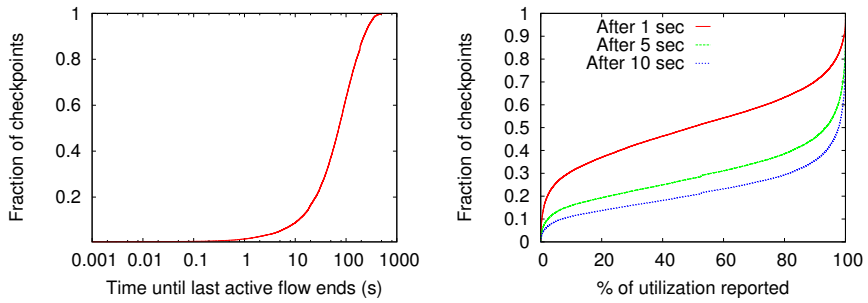

Fig. 3. (left) Distribution of waiting times to compute the total utilization value at every FlowRemoved event. **(right)** Utilization reported after 1s, 5s, and 10s following the expiry of a flow entry. Around 70% of links have 90% or more of total utilization reported after 10 seconds.

age time between FlowRemoved events, under the assumption that all flows arrive on the same link, and find that a flow expires, and thus enables us to refresh the utilization measurements, every 16ms.

In reality, however, flows arrive at a switch on different input ports. Because the traffic trace does not contain input port information, we simulate a 24-port switch using the following heuristic. We first associate every distinct $/p$ prefix (where p is, in turn, 32, 30, 28, 20, or 24) of source IP addresses in the trace with a port and then assign each individual flow to the link (or input port) associated with its source IP $/p$ prefix. We group flows by prefix because routing in the Internet is typically prefix-based. Below, we present results for $p = 28$.

We compute the average time between two consecutive utilization checkpoints for each port and plot the cumulative distribution in Figure 2(right). Here, consider the line labeled "All". For half of the incoming links, the average time between two utilization measurements is at most one second and for almost 90% of the links under 3 seconds. We also performed the heuristic to simulate a 48-port switch with various prefix sizes and obtained similar results.

4.3 Staleness

To compute the total utilization at a checkpoint, FlowSense must wait for all the flows active at the checkpoint to finish and trigger FlowRemoved messages. For each checkpoint, we define the utilization wait time as the time until the last active flow expires. Figure 3(left) shows the cumulative distribution of the utilization wait times for each checkpoint in the trace described in Section 4.2, where flows are assigned to one of 24 incoming links. The median utilization wait time is 98s: for almost half of the checkpoints, FlowSense would have to wait more than 100s to capture the complete utilization.

The long delay in computing the total utilization may be caused by active flows that are very long but do not send a lot of traffic (*e.g.*, ssh sessions). Next, we show that if an application is willing to tradeoff some accuracy for timeliness, it can have a reasonable estimate of a link's utilization at a particular checkpoint in under 10s, rather than

having to wait for 100s. We compute how much of the total utilization at a checkpoint is reported by FlowSense 1s, 5s, and 10s after the checkpoint is created. Figure 3(right) shows that FlowSense reports around 60% of the total utilization for 50% of the checkpoints after 1s, and 90% of the total utilization for 70% of the checkpoints after 10s.

The granularity of measurements does not decrease by much when considering only the 70% of checkpoints that capture 90% after 10s. The line labeled "90% util reported" in Figure 2(right) shows the distribution of the average time between these checkpoints. The median time is only around 1.7s (increasing from 1.1s when considering all checkpoints).

To summarize, FlowSense is able to refresh utilization less than every 2s on average and obtain 90% of the total utilization at these refresh checkpoints in under 10s. We are investigating ways to predict the utilization wait time at each checkpoint. Such a prediction would give applications another knob to tune measurement performance: if the wait time is too high, the application could decide to trigger on-demand polling, thus trading off scalability for lower measurement staleness.

5 Discussion

We designed FlowSense to work for reactive OpenFlow deployments, where switches trigger control messages every time a new flow arrives or a flow entry expires. The presence of a large number of flows triggers many control packets and can overwhelm both the controller, which cannot process all control traffic in a timely fashion, and the switches, which cannot operate at line speed and quickly exhaust their flow tables [12]. Previous research shows that such deployments are feasible for medium-sized networks with a powerful controller or a collection of controllers. For example, controllers in networks of 100 switches, with new flows arriving every 10μs, may have to process up to 10 million PacketIn messages per second [2].

In practice, the need for scalability pushes operators to increasingly adopt alternative OpenFlow deployments: distribute controller functionality across different machines, set up rules proactively to never expire (*e.g.*, with infinite timeouts) so as to avoid triggering control traffic, and use wildcard rules to reduce the amount of control traffic. We discuss next the applicability of FlowSense in such scenarios.

Distributing the controller. Distributing the controller does not affect the amount or frequency of control traffic. Using a mechanism similar to FlowVisor [9], FlowSense could still capture incoming control traffic and synchronize the information gathered across controllers.

Proactive rules and large timeouts. When operators install rules proactively, new flows at a switch do not trigger PacketIn's because they find a matching rule in the flow table. Further, if rules have large timeouts, they take long to expire and trigger FlowRemoved's. Some entries may even be set up to never expire or to not trigger a FlowRemoved when they expire. In such scenarios, control traffic is scarce or missing completely and polling switch counters for utilization provides more frequent utilization estimates, albeit at the expense of network overhead. For reactive applications that rely on traffic changes, they will have to either rely on stale data or begin active polling as previously stated.

Wildcard rules. Wildcard rules limit the number of FlowRemoved messages and forces us to resort to active solutions such as polling counters more often. More importantly, certain wildcard rules can make the utilization computation impossible. If a rule has a wildcard for the input port then the rule is not associated with a single link. Thus, we cannot infer how the traffic that matches against the rule is divided among the input ports to which the wildcard refers to and we cannot compute utilization on the links that end in these input ports.

6 Conclusions

We presented FlowSense, a tool to efficiently infer link utilization in flow-based networks by capturing and analyzing control messages between switches and the controller. Using experiments on a small OpenFlow testbed and simulations on a traffic trace from a campus network, we showed that our method is accurate and provides up-to-date information when control messages are abundant. Our work is the prelude to a larger research direction that we intend to explore in the future: how can we leverage information carried on the control channel of flow-based networks, that is unavailable in traditional networks, to build more robust and accurate monitoring systems and tools.

References

1. J. R. Ballard, I. Rae, and A. Akella. Extensible and scalable network monitoring using OpenSAFE. In *INM/WREN*, 2010.
2. T. Benson, A. Akella, and D. Maltz. Network traffic characteristics of data centers in the wild. In *ACM IMC*, 2010.
3. Z. Cai, A. L. Cox, and T. E. Ng. Maestro: A System for Scalable OpenFlow Control. Technical Report TR11-07, Rice University, 2011.
4. Genesis Hosting Solutions. http://www.nec.com/en/case/genesis/index.html.
5. IBM and NEC team up. http://www-03.ibm.com/press/us/en/pressrelease/36566.wss.
6. L. Jose, M. Yu, and J. Rexford. Online measurement of large traffic aggregates on commodity switches. In *USENIX Hot-ICE*, 2011.
7. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *ACM Sigcomm CCR*, 38:69–74, March 2008.
8. Selerity. http://seleritycorp.com/.
9. R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the test-bed. In *USENIX OSDI*, 2010.
10. Tervela. http://www.tervela.com/.
11. A. Tootoonchian, M. Ghobadi, and Y. Ganjali. OpenTM: Traffic Matrix Estimator for OpenFlow Networks. In *PAM*, 2010.
12. M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable flow-based networking with DIFANE. In *ACM Sigcomm*, 2010.