QUARTERLY TECHNICAL PROGRESS REPORT

Shakedown Experimentations and Prototype Services on Scalable, Agile, Robust, and Secure Multi-Domain Software Defined Networks

Report Period: Apr. 1, 2015 – Jun. 30, 2015

Technical Point of Contact
Professor Chen-Nee Chuah, Professor Matt Bishop, Professor S. J. Ben Yoo
University of California
Dept. of Electrical and Computer Engineering
Kemper Hall, Rm 3179
Davis, California 95616
Tel: 530-752-7063
Fax: 530-752-8428
E-mail: chuah@ucdavis.edu, bishop@ucdavis.edu, sbyoo@ucdavis.edu

Table of Contents

a) Major Accomplishments

1. Milestones Achieved

Table 1 summarizes the status of completion for the different milestones indicated in Year 2 period. This report discusses in particular the technical progress related to tasks and milestones for the period April 1, 2015 – June 30, 2015.

Table 1. List of milestones achieved with status of completion.

Task	Milestones	Status
1	Demonstration of how close-loop analysis, programmable measurement, and traffic inferences using OpenFlow can be applied to detect distributed/global icebergs. Multiple OpenFlow controllers will run iSTAMP modules to adapt the traffic measurement rules on OpenFlow switches dynamically.	COMPLETED
2	Use GENI resources including GENI OpenFlow switches and virtual machines for the demo	COMPLETED

The following sections will describe in details the studies and finding related to the tasks mentioned above. In particular, we focused on applying SDN in the application of network tomography and global iceberg detection. The following sections will describe in details the studies and finds related to each of the activities above.

2. Deliverables Made

The deliverable include:

3. A poster has been presented in GEC'23, showing the project progress and our demos.
4. A live demo showing software-defined networking enabled traffic matrix estimation and global iceberg detection on GENI testbed has been successfully presented in GEC'23.

b) Description of Work Performed During Last Quarter

1. Activities and findings

We proposed a new framework for network tomography which adopted software-defined networking (SDN) enabled online learning. In our proposed SDN measurement framework as shown in Figure 1, multiple SDN switches are distributed in the network. A centralized controller could communicate with all of the SDN switches. In control plane, the controller could: 1) fetch flow statistics from SDN switches, 2) update the measurement rules online. The centralized controller has a network-wide view of the global routing. We consider the case where these SDN switches perform both packet forwarding as well as measurement tasks, so that we could avoid additional measurement hardware deployed on top of traditional routers to reduce the implementation cost and complexity in practical cases. Thus for each SDN switch, part of the TCAM entries are pre-populated by local routing rules and can not be altered in order to preserve routing. In this way, we could get the aggregated flow statistics from these routing entries for free.

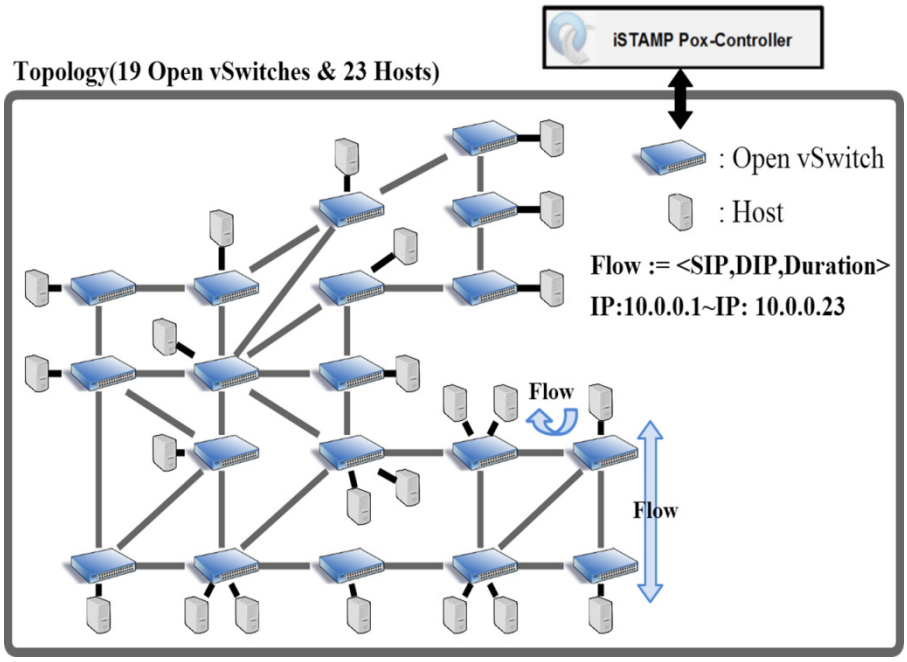

Figure 1 Overview of SDN-based network

Besides routing entries, a handful of TCAMs are available to implement measurement rules. To improve estimation accuracy, new measurement rules can be generated by offloading subsets of flows matching the original routing rules. However due to the limitation of the current implementation of TCAMs (only flows with source/destination IPs belonging to the same IP subsets can be aggregated in one TCAM entry), these subsets of flows can not be arbitrarily selected. In our framework, we implement per-flow measurements in the available TCAMs to directly measure the most important flows (i.e. flows with largest flow size). The idea is that 1) directly measuring the most important flows could provide higher estimation accuracy improvement compared to measuring randomly selected flows. 2) per-flow measurements are feasible in practical implementation and routing function is easy to be preserved.

To find the most important flows, we apply an intelligent learning algorithm (MUCB) to rank all the flows based on the historical flow size information and estimate the location of the large flows in the next time interval. By utilizing the real-time reconfiguration feasibility provided by SDN, the controller will update the measurement rules periodically to track the most important flows. However, since the measurement resources are distributed in the network, it need to find a feasible solution to allocate "flow measurement rules" to the distributed SDN switches with available measuring TCAMs. We proposed two heuristic solutions for the "flow-measurement-rules allocation" problem.

The first algorithm is called "LastHop", whose pseudo code is shown in Figure 2. The second algorithm is called "Greedy", whose pseudo code is shown in Figure 3. In both heuristic solutions, we will first rank all the flows in the network in decreasing order of "importance" estimated by MUCB and then start fill the available TCAMs with measuring rules for flows with higher "importance". Lasthop algorithm will first try to measure a selected flow at the last SDN switch before reaching destination. If there is no TCAMs available at the current, the algorithm will then check if the prior hop is feasible to measure it. Continue this process until we find a spot to install the per-flow measurement for the selected flow or no switches are available to measure it. The "Lasthop" algorithm will exit once all the available measuring entries are filled up with measuring rules. On the other hand, Greedy algorithm will list all the available SDN switches that a selected flow goes through in its flow path. It will install the measurement rule at the switch which has the most number of available TCAM entries up-to-date.

To summarize, the centralized controller will: 1) fetch flow statistics (both routing statistics and measurement statistics) periodically, 2) estimate flow size for all the flows in the network, 3) apply MUCB algorithm to sample the most important flows to be directly measured in the next time interval, 4) run heuristic algorithms to find a feasible solution to allocate "flow-measurement rules" to the distributed SDN switches with available TCAM entries. 5) update the measurement rules according to the output of step 4).

Algorithm 1 Lasthop

```
Input: index=[ranking flows based on their "importance"], distribution of
measuring entries, assume k_j TCAM entries available at switch j
Output:a feasible switch-flow allocation
for each flow fl in index do
    switch_list = [switches flow fl goes through from dst to src]
    for each switch j in switch_list do
        if k_j > 0 then
            install per-flow measurement for flow fl here
            k_j = k_j − 1
            break
        end if
    end for
end for
```

Figure 2 Pseudo code of Lasthop Algorithm

Algorithm 1 Greedy

```
Input: index=[ranking flows based on their "importance"]; distribution of
measuring entries, assume k_j TCAM entries available at switch j; load for
each switch: number of important flows passes trough this switch
Output: a feasible switch-flow allocation
for each flow fl in index do
    switch_list = [switches flow fl goes through from dst to src]
    choose switch j ∈ switch_list where k_j is largest
    if there is a tie then
        choose switch j which has the least load
    end if
    if k_j > 0 then
        install per-flow measurement for flow fl here
        k_j = k_j − 1
        break
    end if
end for
```

Figure 3 Pseudo code of Greedy Algorithm

In 23rd GENI Engineering Conference (GEC'23) we demonstrated an implementation of our network measurement framework on GENI testbed as the following:

(1) Practical Implementation and demo

We demonstrated an implementation of our network measurement framework on GENI testbed. We simulated our framework using GEANT network topology, which has 23 switches and 37 links. We used the jFed, a Java-based slice reservation tool, to reserve our topology. We created 46 slices and 60 links from Kettering InstaGENI aggregate as shown in Figure 4. 23 slices are used as traffic generators to inject traffic flows into our simulator. We used real traffic traces of GENAT network in our simulation. Open vSwitch was installed and configured on the other 23 slices to simulate the function of OpenFlow switches. Another VM with a public IP address was reserved to run our OpenFlow controller.

In the demo, network flow generators are used to generate network flows according to the flow trace file collected in GEANT network. The generated flows are injected into the SDN switches. The SDN switches forward and count the flows according to the flow rules installed in the flow table, and the SDN switches also reply to the flow statistics request sent by the centralized traffic measurement controller. The traffic measurement controller periodically get flow statistics from the SDN switches, and estimate individual flow sizes at each time interval using those statistics. An learning algorithm is running at the controller side to predict the locations of large flows for the next interval. The controller then runs "Greedy" or "LastHop" algorithm to allocate those selected flow-measurement rules to SDN switches, and update the flow tables at switch side accordingly. The block diagram in Figure 5 shows the interactions between the centralized controller and SDN switches in detail.

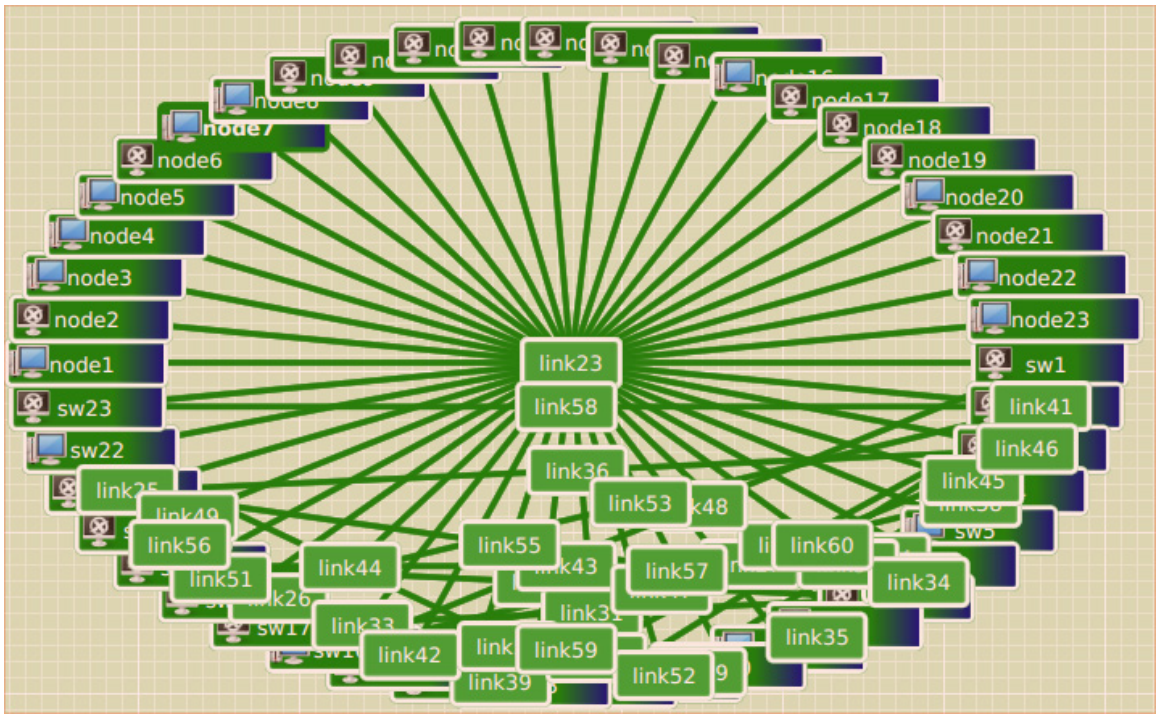

Figure 4 The Overview of GEANT topology on GENI testbed

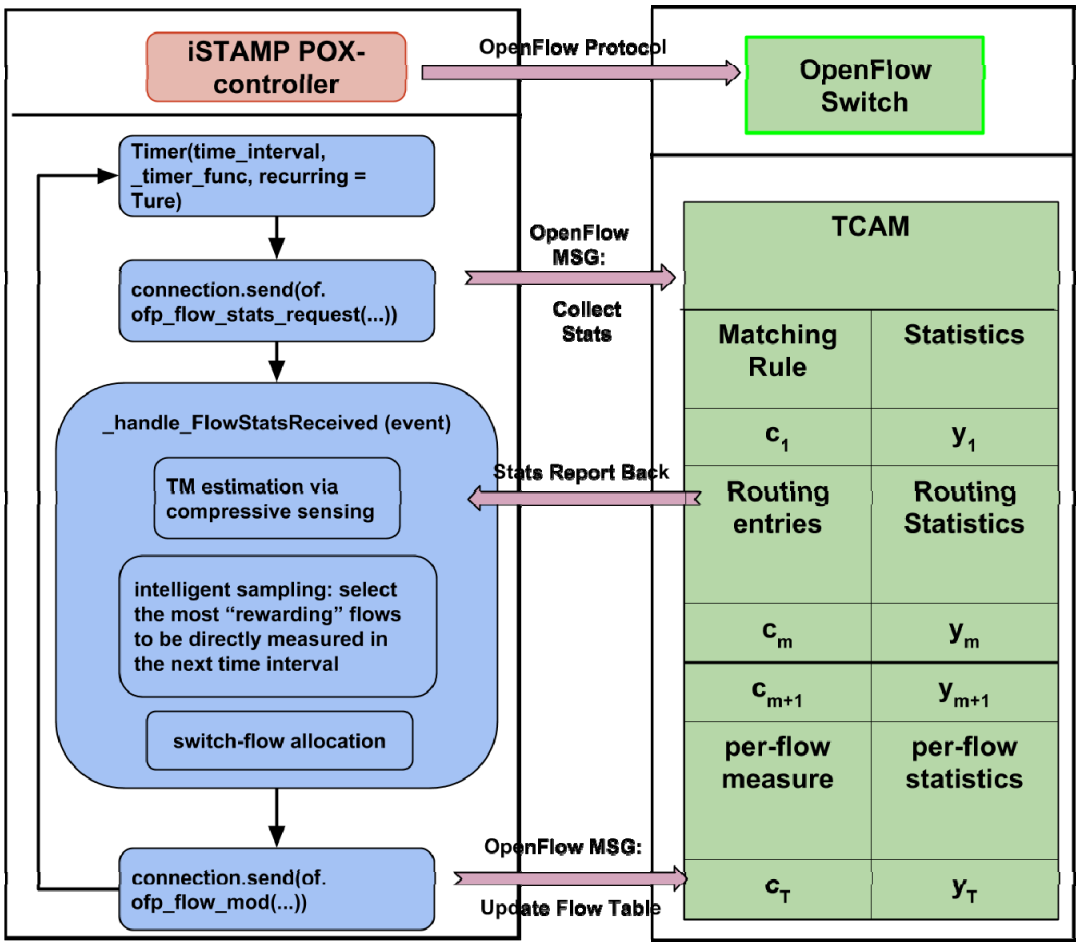

Figure 5 Block diagram of the interactions between controller and SDN switches

The GUI of the demo is shown in Figure 6. There are two windows in the GUI. In the first window, plot2 shows the estimated flow sizes (blue) and real flow sizes (red) of all flows in the network. Plot1 tracks a particular flow (both estimated flow size in solid blue line and real flow size in dashed red line) in real time. Performance metrics, NMSE and HH detection, are also shown in this window. The second window shows the Hierarchical Heavy Hitter (HHH) detection in real time. To detect HHH, we build a prefix tree of source IP addresses for each destination as shown in Figure 7. Nodes shaded with green color are true HHHs which are detected by our framework. Nodes shaded with blue color are true HHs which are detected by our framework. Nodes shaded with red color are false alarms. The user can select which destination to monitor on the panel by using the drop-down list below.

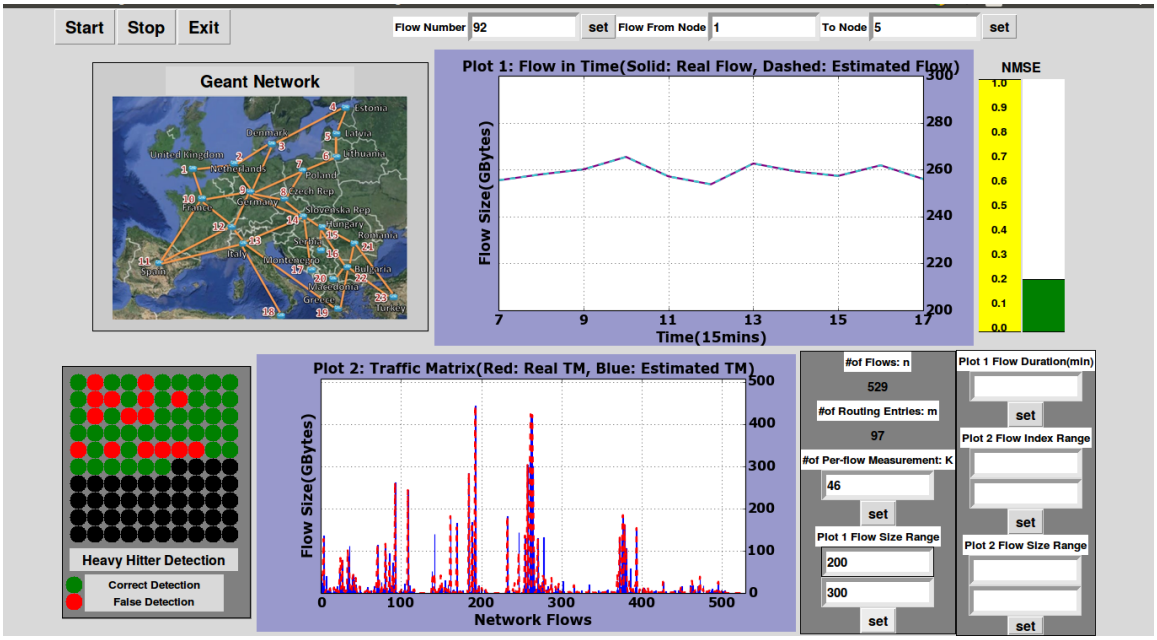

Figure 6 The GUI of the demo (window #1)

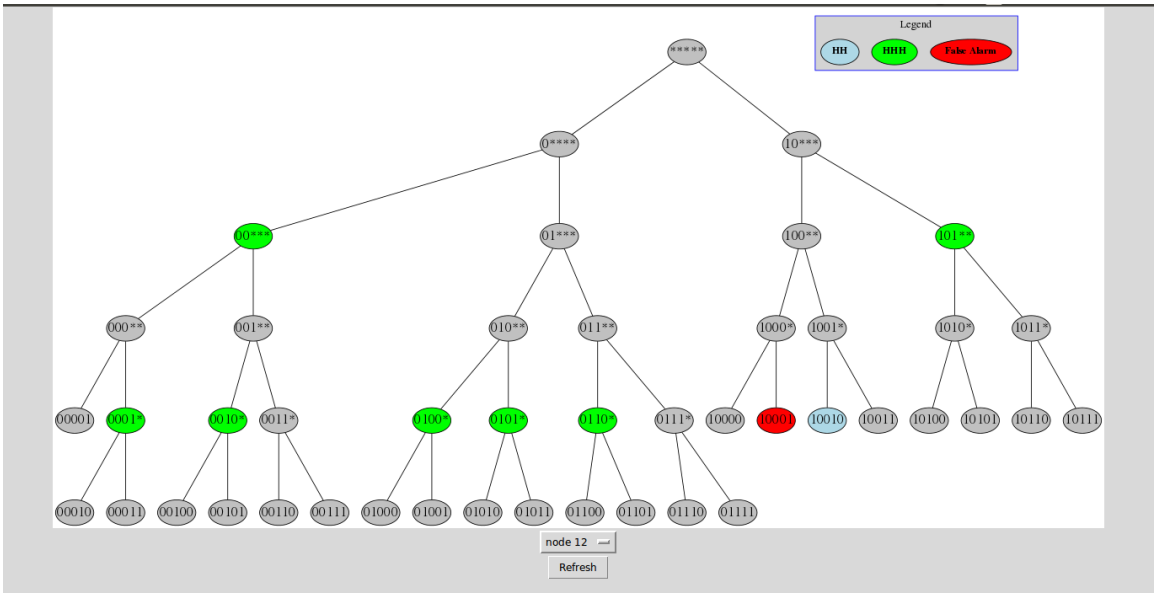

Figure 7 The GUI of the demo (window #2)

(2) Simulation Result

We evaluate our framework using the metric NMSE, normalized mean square error, which is a widely used performance metric for measuring the accuracy of traffic matrix estimation. We also evaluated the performance of our framework in the application of Heavy Hitter (HH) detection and Hierarchical Heavy Hitter (HHH) detection.

Figure 6 shows the NMSE of different methods when the number of available measuring entries of each switch varies. The "random" method will randomly sample flows for direct measurements. As we can see, compared to the "random" method, where flows are randomly

selected for direct measurements, the two algorithms which keep tracking of the largest flows with learning provide a smaller NMSE and thus a higher estimation accuracy. Greedy algorithm outperforms the LastHop algorithm when the number of available measuring entries is small.

A flow is considered to be a Heavy Hitter if the flow size is larger than a threshold. To justify the effectiveness of our framework for HH detection, the average probability of detection (P^{d}_{HH}) and the average probability of false alarm (P^{fa}_{HH}) are used. Figure 7 and Figure 8 show that we could get a higher detection rate and a lower false alarm rate when equipped with the learning feasibility provided by SDN compared to the random sampling.

To detect Hierarchical Heavy Hitters (HHHs), we build a prefix tree of source IPs for each destination. An illustrative example is shown in Figure 5. An aggregated prefix node is considered to be an HHH if its aggregated flow size, excluding the flow size of its HHH descendants, is larger than a threshold. In this example, the threshold is set to 10. Nodes in double circles are HHs and nodes in shaded circles are HHHs. We adopt the two performance metric, recall and precision to quantify the performance of our framework. Recall is defined as the total number of true HHHs detected over the real number of HHHs. Precision is defined as the total number of true HHHs detected over the total number of HHHs reported. Figure 9 and Figure 10 show that a much higher recall and a much higher precision could be achieved when applying SDN-enabled learning. Compare the two resource allocation algorithm, we could see that Greedy is better than LastHop. This is reasonable since we choose wiser in allocating flow-measurement rules to available switches.

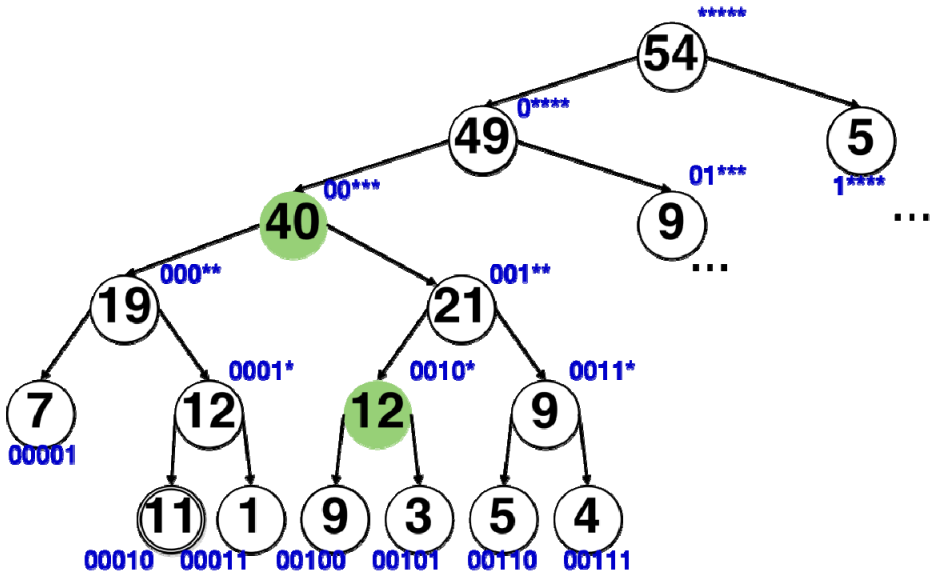

Figure 6 An illustrative example of an IP prefix tree

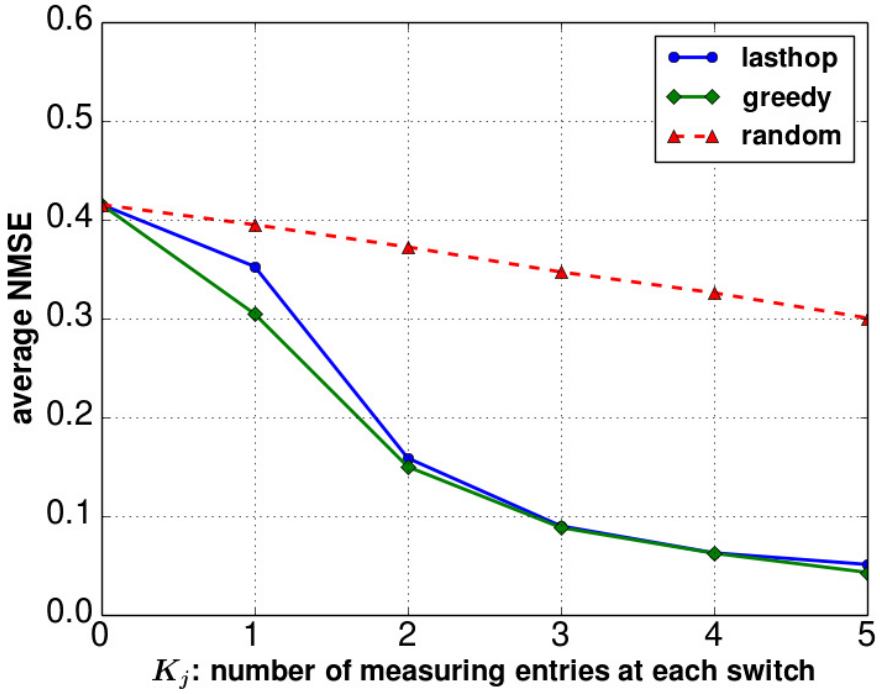

Figure 6 NMSE in GEANT when K_j varies

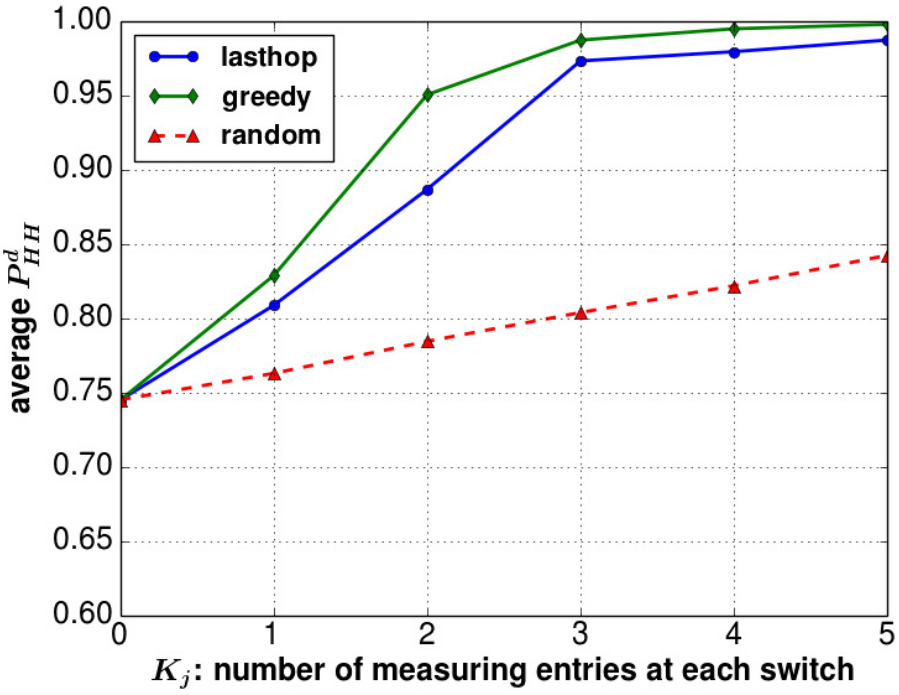

Figure 7 Probability of detection when K_j varies

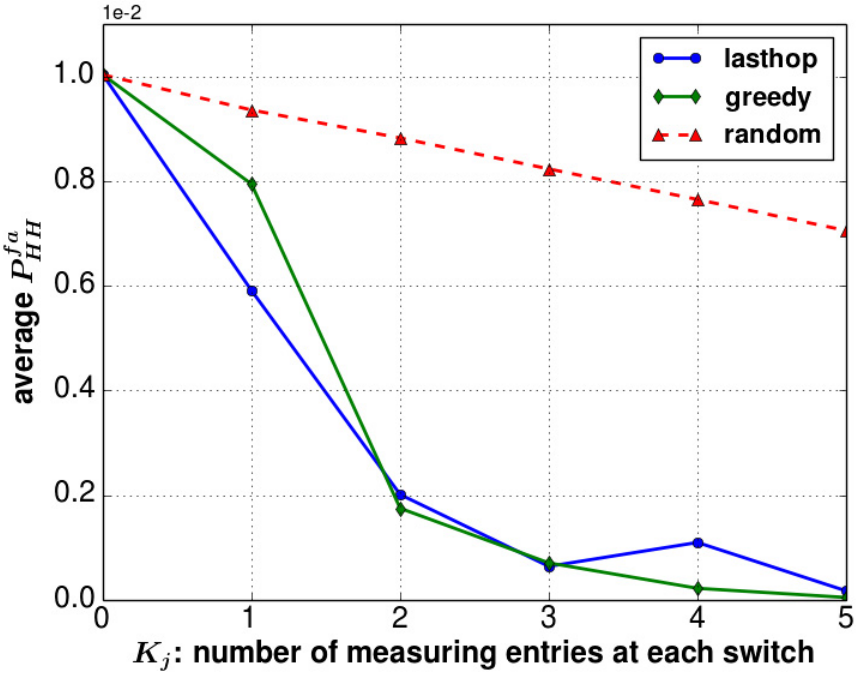

Figure 8 Probability of false alarm when K_j varies

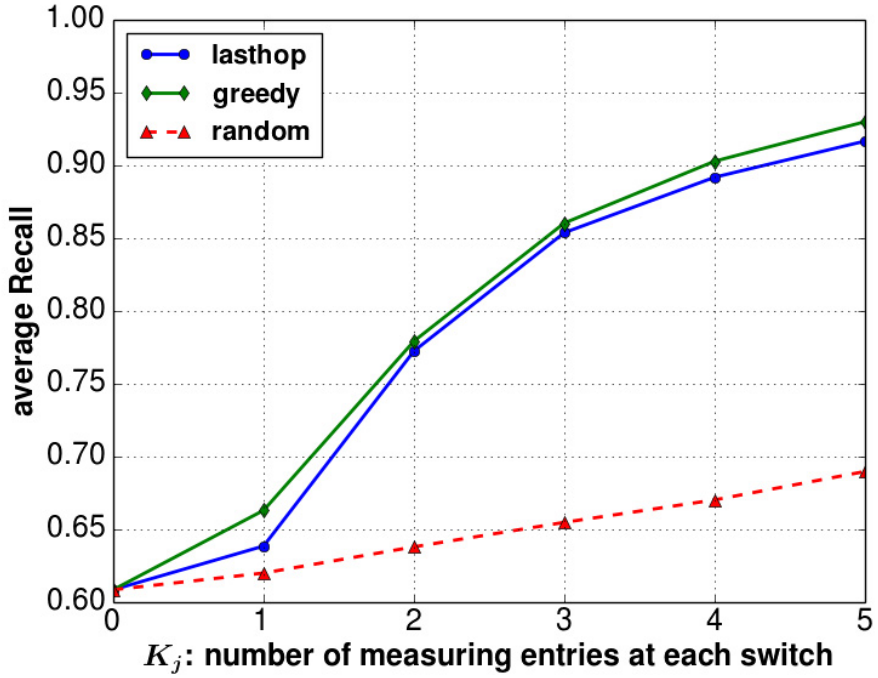

Figure 9 Average Recall when K_j varies

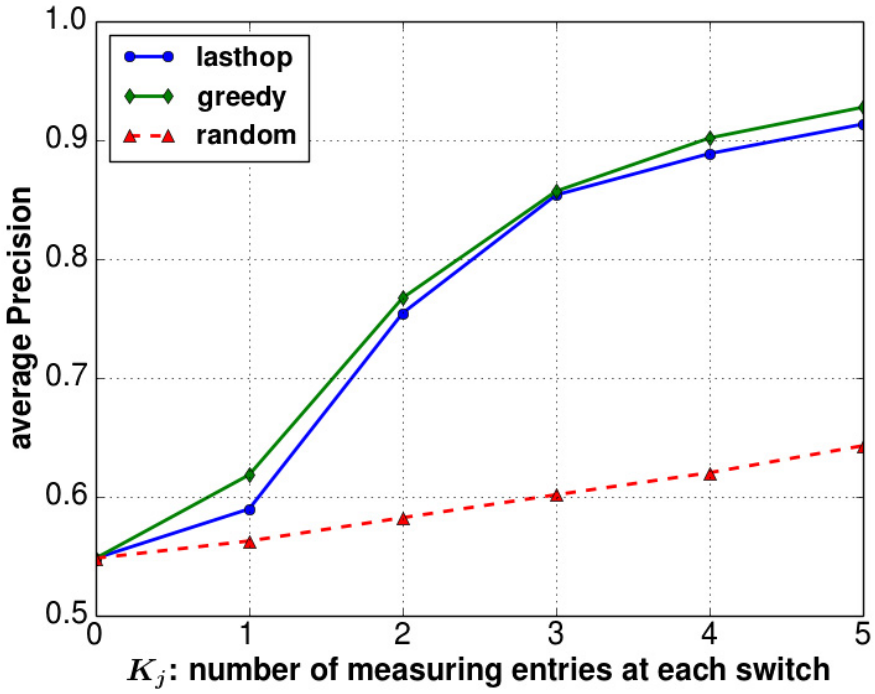

Figure 10 average Precision when K_j varies

2. *Project participants*

Prof. S. J. Ben Yoo	*Heterogeneous Multi-Domain Network Testbed*	UC Davis, PI
Prof. Matt Bishop	*Security in Scalable Programmable Networks*	UC Davis, Co-PI
Prof. Chen-Nee Chuah	*Monitoring in Scalable Software Defined Networks*	UC Davis, Co-PI
Ms. Chang Liu	*Network measurement framework and demo*	UC Davis
Ms. Shu Ming Peng	*Mininet*	UC Davis
Mr. Mehdi Malboubi	*Network measurement framework*	UC Davis

3. *Publications (individual and organizational)*

[1] M. Malboubi, L. Wang, C-N. Chuah, and P. Sharma, "Intelligent SDN based Traffic (de)Aggregation and Measurement Paradigm (iSTAMP)," IEEE INFOCOM, April/May 2014

4. *Outreach activities*

N/A

5. *Collaborations*

N/A

6. *Other Contributions and Future Plans*

N/A