

Real-time End-to-end Network Monitoring in Large Distributed Systems

(Invited Paper)

Han Hee Song
University of Texas at Austin
Austin, TX, USA
hhsong@cs.utexas.edu

Praveen Yalagandula
HP Labs
Palo Alto, CA, USA
praveen.yalagandula@hp.com

Abstract—

Measuring real-time end-to-end network path performance metrics is important for several distributed applications such as media streaming systems (e.g., for switching to paths with higher bandwidth and lower jitter) and content distribution systems (e.g., for selecting servers with lower latency). However, it is challenging to perform such end-to-end pairwise measurements in large distributed systems while achieving high accuracy and avoid interfering with existing traffic. On the end hosts, the measurements can overload the machine by causing interference among themselves and other processes. On the network, the measurement packets from different hosts can interfere among themselves and with other flows on bottleneck links. In this paper, we propose a system to monitor end-host and network resources and adapt the number of measurements according to the observed load. Our scheme avoids interference by measuring only a small subset of network paths and reconstructing the entire network path properties from the partial, indirect measurements. Our simulation experiments and real testbed experiments on PlanetLab show that our path selection algorithm working with resource constraints does not adversely affect the accuracy of inference and our system can effectively adapt to the changing resource usage at the end hosts.

I. INTRODUCTION

Measuring real-time end-to-end network path performance metrics is important for several emerging distributed applications such as content distribution and file sharing systems (e.g., CoDeeN [1], BitTorrent [2], and KaZaA [3]), interactive media streaming systems (e.g., Video conference systems such as HP's Halo [4] and Cisco's Telepresence [5]), traffic engineering (e.g., for supporting IP Telephony [6]), and overlay networks (e.g., RON [7] and OpenDHT [8]). For example, media streaming systems need the network performance information for switching video and audio sessions to better paths with higher bandwidth, lower latency, and lower jitter. Similarly, content distribution systems need network state for selecting servers with low latency to clients to reduce the content transfer time. The goal of the S³ [9] (Scalable Sensing Service) project is to provide a middleware to measure real-time end-to-end network properties at rates as specified by applications in a scalable and efficient manner. One of the challenges for S³ is to support applications such as traffic engineering and ISP performance monitoring systems that require measuring characteristics of all network paths which grows quadratically with the size of the network.

The brute force way of simultaneously measuring all network paths, from each end host to all other end hosts, in a large networked system can lead to inaccurate measurement results and pose an enormous load on node and network resources. In Section II, we present our experiments with different network measurement tools and observe that few tens of simultaneous bandwidth measurements can use up to 50% of CPU and 40% of memory on end hosts and 7Mbps of network bandwidth on a link. A naive scheduling solution that schedules few measurements at a time on each node can reduce the loads on node and network and avoid interference with other processes and network flows. But, this can lead to an increase in the time to measure all paths resulting in stale measurement values and violation of the real-time requirements. In our deployment of S³ [10] on PlanetLab [11], we observe an average of 31 hours for a measurement cycle to complete when we serialize the measurements from each host (refer to Section II).

Inference is a solution to achieve scalability in monitoring large networked systems. Instead of measuring on all paths between end nodes, inference based solutions require few measurements from each node and infer all-pair properties from those few measured paths. There is a large body of research effort focusing on latency estimation [12], [13], [14], [15], [16], [17], [18], loss rate [19], [20], and few on bandwidth [21], [22]. Though the previous researches focus on scalability, they do not consider the resource constraints to decide the set of measurements. For example, in landmark based approaches for latency estimation such as GNP and Netvigator, each node performs latency measurements to a small set of landmark nodes and infers the all-pair path latencies using that information. If a larger set of landmark nodes are used, the accuracy of the estimation will be higher but can incur higher load on the network and end nodes. Also as the size of the node set grows, landmark nodes can get overloaded. This raises a need for inference algorithms that are resource constraint aware and can adapt the paths chosen for monitoring thus providing a trade-off between accuracy and the load.

To address these challenges, we propose to monitor the resources available and determine a set of paths to continuously measure such that they use only a fraction of available resources and that we can accurately estimate all-

pair path properties from the measured information. To be more concrete, in a network of n end hosts with $O(n^2)$ paths, we select a subset of paths S to monitor actively such that we can infer the values of remaining paths while satisfying the following resource constraints – the load incurred due to these measurements is smaller than a fraction f_{node} of free resources at the end nodes and the network load incurred by the measurements on a network hop/link is smaller than a fraction f_{ntwk} of available bandwidth on the link. Since the resource availability at end nodes and bandwidth change with time, we need to constantly monitor these metrics and adapt the set of paths selected to satisfy the invariants on resource usage.

The algorithm for selecting a subset of paths that maximizes the accuracy of inferred values largely depends on the inference technique to be used. Towards our goal, we have started with the loss and latency metrics and focus on leveraging and extending the NetQuest [20] framework. The original NetQuest system proposes scheduling and inferencing algorithms for loss rate and latency estimation but focuses on minimizing the total number of paths used. In this work, we propose a new scheduling algorithm that decides the set of paths to measure based on the end-node resource constraints. Though the resource constraints on network can also be posed in our new algorithm, we currently use end-node free CPU to decide the constraints as the loss rate tool inflicts insignificant load on the network links even when several measurements are running simultaneously (refer to Section II).

Our paper makes several key contributions. First, we motivate the need for resource aware scalable inference algorithms for monitoring large scale networks in real time. Second, we characterize the resource requirements of different network measurement tools. Third, we present an algorithm to decide subset of paths to monitor that satisfy resource constraints while achieving high accuracy in the inference. We have built a prototype of our solution and evaluated it via simulations with data collected from PlanetLab and via deployment on PlanetLab infrastructure. Our system uses S³ deployment on PlanetLab to collect the topological structure of the network between nodes. We run a process on each end node to characterize the loss rate tool, determine how many instances can be run simultaneously without consuming significant portion of the free CPU, and report that number to a central machine. Our central server runs the path selection algorithm and schedules measurements on those selected paths. Our results show that our scheme can accurately estimate individual path properties while using only a small fraction of free CPU capacity at each measurement host.

This initial study focuses on using only one kind of latency and loss rate estimation technique to address our problem of adapting the measurements to perceived load on nodes and network. We are further exploring on tailoring other inference algorithms—both centralized and distributed—to solve our problem. We are also exploring the much harder problem of measuring available bandwidth and capacity.

In Section II, we discuss the motivation for our work in

detail. We describe our approach that leverages and extends NetQuest in Section III. We present evaluation results in Section IV including results from simulation experiments with data from S³ deployment on PlanetLab and results running our system on PlanetLab. We present related work in Section V and conclude with summary and future work in Section VI.

II. MOTIVATION

In this section, we discuss the motivation for our work in more detail. We first explore the impact of running several measurements simultaneously from a single host to multiple hosts focusing on the load posed on the end-hosts and network and also on the interference leading to incorrect results. We then discuss the problems with serialized approach, where measurements from each node are performed in a round-robin fashion. Then we discuss the inferencing as a promising approach to reduce the load on end-hosts and network and discuss how current research in inferencing ignores the state of available resources to decide the schedule of measurements.

A. Impact of Concurrent Measurements

On a host, concurrent measurement processes can use up a significant portion of CPU and memory leading to interference among the measurement processes and can also interfere with other processes on the machine. Also as many end-to-end network paths might share a network link, packets generated by the measurements passing through such shared link can interfere with other traffic and among themselves leading to inaccurate measurement results.

We have experimentally studied the impact of performing concurrent measurements with several network measurement tools. Here we present the results from our experiments with the following tools.

- **LossDelay**: This tool measures round-trip loss rate and latency on a path using *Ping* tool to send 100 packets at a rate of 5 packets per second (maximum rate allowed by Ping at the user level in Linux by default). We report the number of packets lost as the loss rate and report the average round-trip time reported by Ping as the latency.
- **Pathrate** [23]: This tool measures one-way bottleneck capacity on a path. This tool needs access to the both ends of a path.
- **PathChirp** [24]: PathChirp is a tool to measure one-way available bandwidth on a path. Similar to Pathrate, this tool needs access at the both ends of a path.

We perform our measurements on Emulab network testbed [25] where we run multiple instances of each tool on two directly connected hosts. Both end hosts are 600MHz Pentium-3 machines with 256MB memory running Fedora Core Update 4. Connecting the hosts is a duplex link with 100Mbps bandwidth and 2ms latency. For each tool, we create a script to repeatedly invoke the tool to perform a measurement to a specified destination and record the measured value, time taken for the measurement, and any other exceptions reported by the tool. We perform our experiments for each tool separately. For a tool, we start with one process running

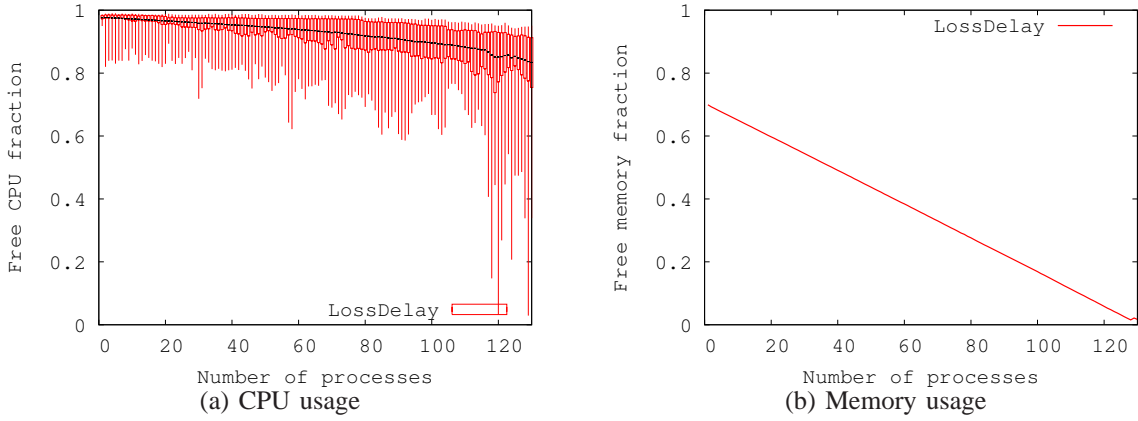


Fig. 1. Resource usage of LossDelay sensor

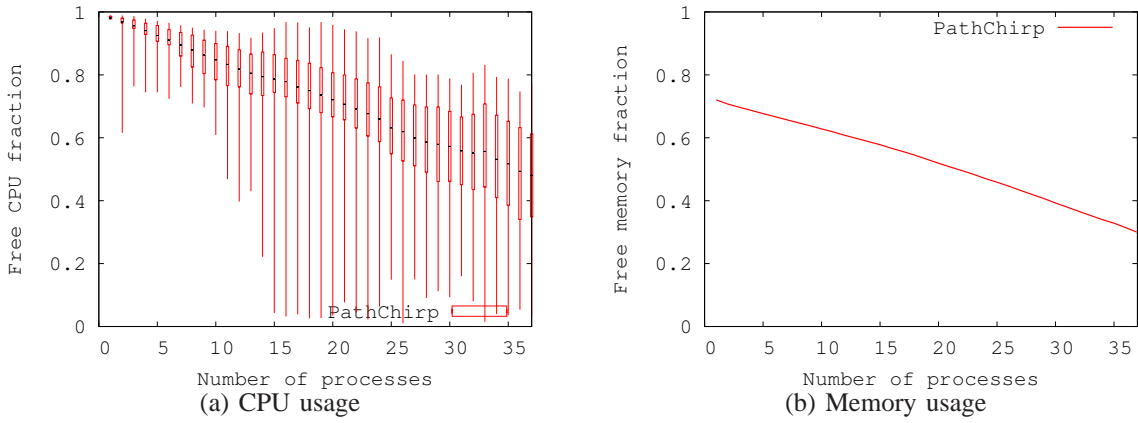


Fig. 2. Resource usage of PathChirp

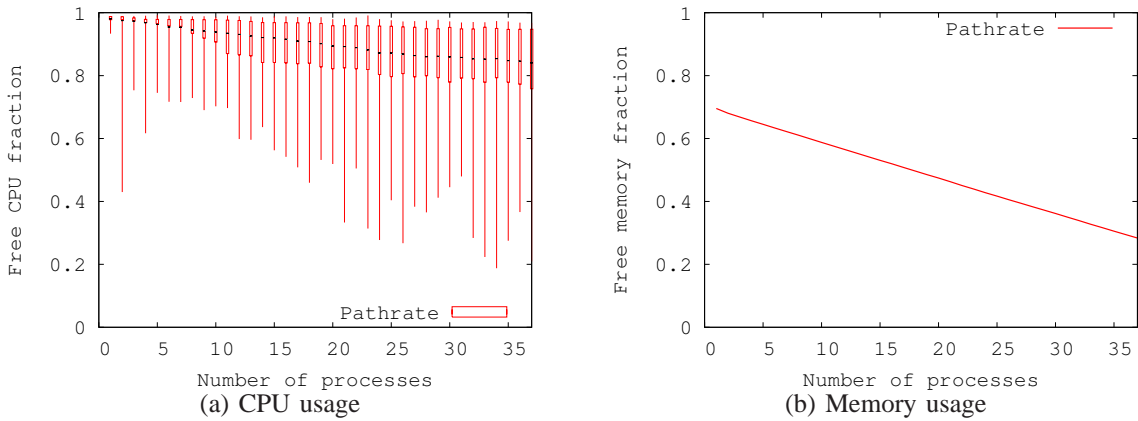


Fig. 3. Resource usage of Pathrate

the script for that tool and every few minutes create a new process running the script to gradually increase the number of concurrent measurements. We choose the period for addition of a new process for a tool depending on the typical comple-

tion times of a single measurement. In our experiments, the period for LossDelay is 2 minutes and the period for Pathrate and PathChirp is 5 minutes.

Resource usage: For each tool, we study the CPU, memory,

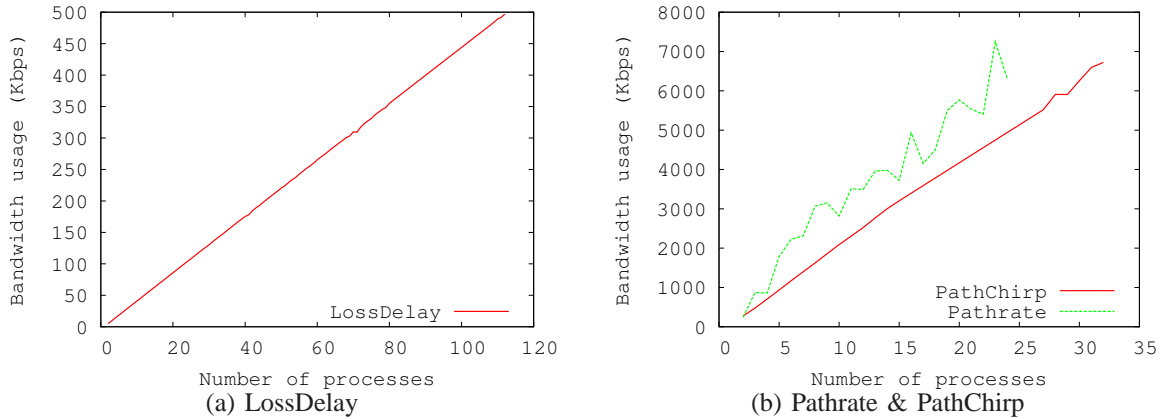


Fig. 4. Network usage of different measurement tools.

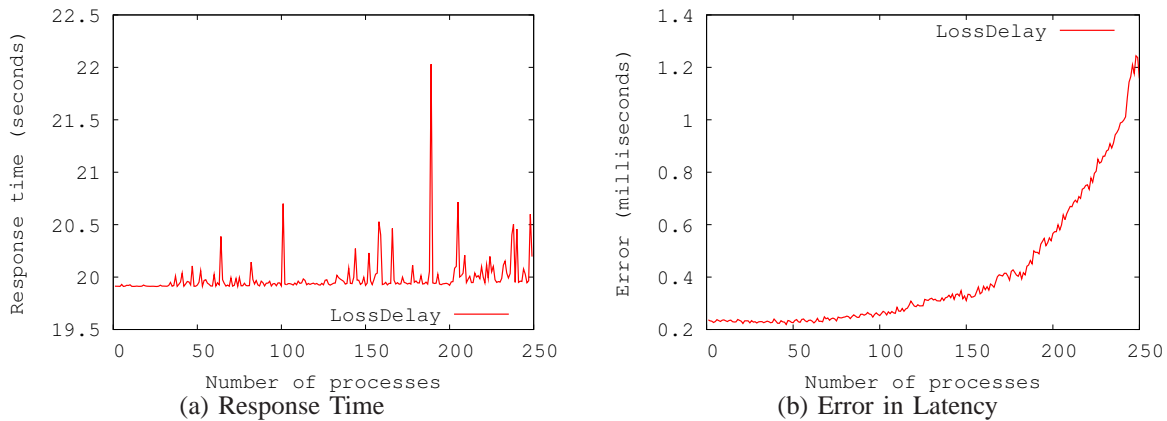


Fig. 5. Interference due to running several concurrent LossDelay measurements on a path

and network bandwidth requirements as we increase the number of concurrent measurements. For CPU, we periodically monitor the `/proc/stat` file to determine the time spent on the idle task and infer the fraction of free CPU from that value. Similarly, for memory, we periodically extract the fraction of free memory from the `/proc/meminfo` file. In Figures 1, 2, and 3, we plot the measurement load observed on a host in terms of the free fraction of CPU and memory as we increase the number of concurrent measurements for LossDelay, PathChirp, and Pathrate respectively. For CPU, we plot the minimum, maximum, 20 percentile, and 80 percentile values along with the average value of the monitored samples when running a set of processes concurrently. For memory, we plot the average values. Though a single LossDelay sensor has very small CPU and memory requirements, few hundreds of concurrent processes consume a significant portion of both CPU and memory. We observe PathChirp to be the most resource intensive tool of all three tools. Even few tens of concurrent PathChirp instances consume more than 50% of CPU and 40% of memory. Pathrate has similar memory requirements as PathChirp but consumes moderate CPU with

tens of measurements.

We plot the measurement load on the access link in terms of the bandwidth used by the tools in Figure 4. As LossDelay sensor sends small sized ICMP packets at a slow pace, the network resource requirement of this tool is very small. A set of hundred concurrent measurements causes only 500 Kbps of traffic. But both bandwidth and capacity measurement tools incur high traffic on the network. Few tens of measurements of either PathChirp or Pathrate consume up to 7Mbps of traffic on a 100Mbps link.

Interference: For understanding the interference caused by concurrent measurements, we analyze the results reported by the measurement tools and the response time of the tools. The response time of a tool is the time taken for a measurement to complete and return a response. In Figure 5, we plot the increase in the response time and inaccuracy of sensor measurements as we increase the number of concurrent measurements. Note that the response time of LossDelay sensor in a normal case is about 20 seconds. As the number of concurrent processes increase, we observe an increase in the response time and also observe an increase in the latency value

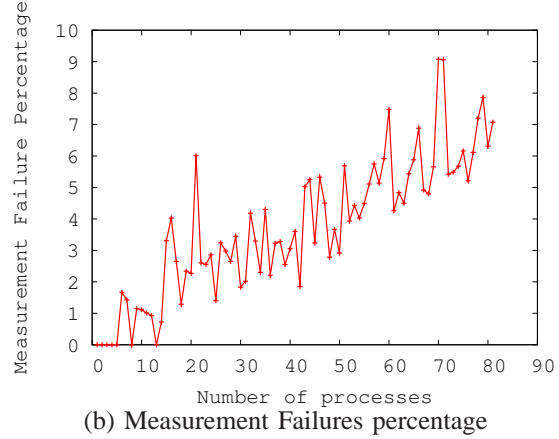
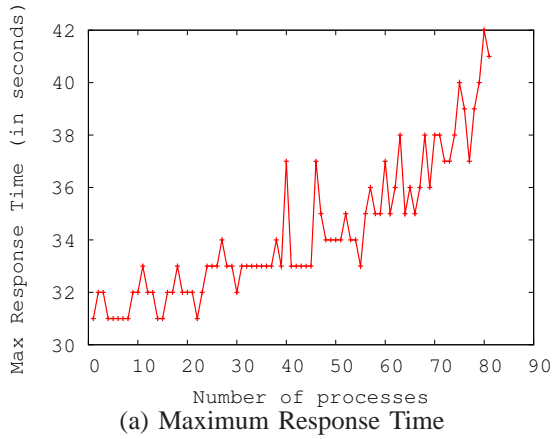


Fig. 6. Interference due to running several concurrent PathChirp measurements for a path. (a) Maximum response time of any measurement where response time is defined as the time taken by tool to complete a single measurement. (b) Percentage of measurements that failed due to interference.

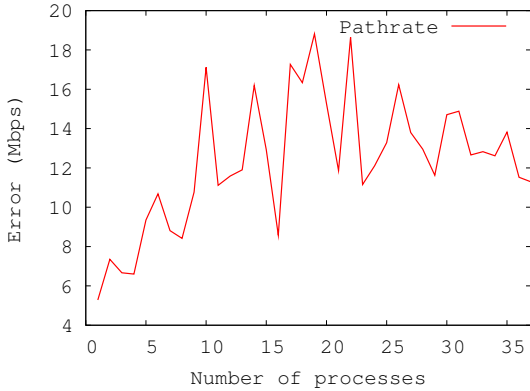


Fig. 7. Interference due to running several concurrent Pathrate measurements from a node. We plot the values returned by the measurements while measuring a 100 Mbps link.

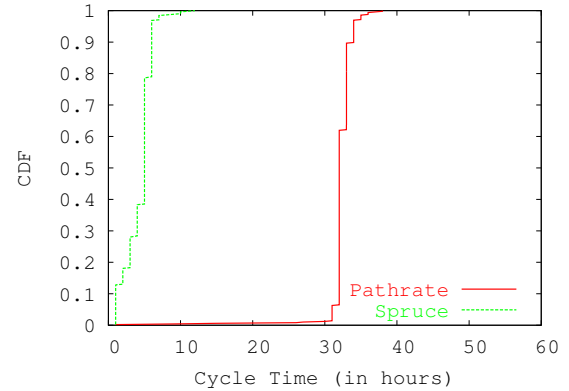


Fig. 8. CDF of times taken for a single cycle of measurements on nodes in S^3 's PlanetLab deployment for two network measurement tools – Pathrate and Spruce.

reported by the LossDelay sensor.

For PathChirp, we plot the maximum response time taken by any measurement in Figure 6(a) and the percentage of failed measurements in Figure 6(b) as we increase the concurrency. Though we did not observe any significant inaccuracy in the results with increasing concurrent measurements (result not included in this paper), we observe significant increase in the percentage of failed measurements where no results are reported. In Figure 7, we plot the average absolute error in the values reported by Pathrate tool as we increase the number of concurrent measurements. The increase in the error shows that the Pathrate's results are sensitive to the interference.

In summary, our experiments quantify the performance of different measurement tools and show that when multiple measurements are performed simultaneously, they (i) consume significant end-host CPU and memory resources and network bandwidth and (ii) suffer from interference leading to inaccuracy in the reported results, increase in failed measurements, or increase in response times.

B. Serializing Measurements

Though scheduling a small number of measurements at a time can reduce the node and network loads and avoid interference with other processes and network flows, it can lead to an increase in the time to measure all paths resulting in stale measurement values and violation of the real-time requirements. In our current deployment of S^3 system on PlanetLab [10], we perform measurements for different network metrics from each node to all other nodes (≈ 500 nodes) in a round-robin fashion. In Figure 8, we plot the CDF of times taken for a single cycle of measurements to complete at nodes in our deployment. We show the CDF for two different tools – Pathrate [23] used for measuring capacity and Spruce [26] used for measuring available bandwidth. Our logs round-off the completion times to the nearest hour, and hence the CDF curves appear as step-functions in this plot. Overall, we observe that a cycle of bottleneck capacity measurements using Pathrate tool takes about 31 hours on average and about 6 hours on average for measuring available bandwidth using Spruce.

Serializing measurements on the sources might not be

enough to combat the overload on resources for certain tools. Bandwidth and capacity measurement tools such as Pathrate, Spruce, and PathChirp [24] need to be run at both source and destination simultaneously and consume significant resources at destination hosts too. Without any coordination between sources, even serialized measurements at sources can lead to a case where several sources might measure the paths to a single destination simultaneously. In our S³ deployment on PlanetLab, even though we randomized the list of destination machines at each node, we still received occasional messages from some nodes where PlanetLab Monitoring process (pl_mom) reset our slice as we consumed a large amount of resources. This happened on a machine when it was the destination for a considerable number of bandwidth measurements from different sources. Note that the time for a bandwidth measurement to complete increases with the load on either source or destination. So, a highly loaded machine on PlanetLab attracts several overlapping measurements from different sources, which in turn increases the load on the machine. This vicious cycle leads to the explosion of processes, exhaustion of swap space, and our slice being reset by the monitoring process.

C. Current Inference Algorithms

Inference techniques are a solution for monitoring large scale networks. Inferencing leverages the fact that many paths share links and hence share the link properties. These techniques typically measure the properties of few paths and reconstruct the properties of all paths from those few measurements.

Researchers had great success with latency estimation as the latency metric is physically tied to the topology and routing decisions which do not change that often for many paths. For example, coordinate-based inference algorithms such as GNP [14] and Vivaldi [12] leverage this fact and assign a global coordinate to each node in a high dimensional space where the distance between two nodes reflects the latency between those nodes in the network. These papers demonstrate the high accuracy of this solution in different settings.

But metrics such as available bandwidth, loss rate, and jitter are highly dynamic that depend on the traffic in the network. Moreover, metrics such as available bandwidth does not respect triangular inequality in contrast to latency metric which has been shown to satisfy triangular inequality in most networks [14], [12], [17]. Hence, it is hard to use coordinate based approaches for these metrics. Techniques proposed for loss rate [27], [19], [20] and available bandwidth [21] either use landmarks to which all nodes perform measurements or decide a subset of paths to measure. But these techniques do not consider resource constraints either at the end hosts or on the network links.

III. APPROACH

The key four tasks involved in our system are as follows: (1) Characterize measurement tool resource requirements, (2) Monitor available resources, (3) Select a set of paths to

measure, and (4) Measure the selected paths and infer the metric for the remaining paths. Note that the first step is a one-time task that we perform before beginning the continuous execution of the later steps.

We leverage and extend NetQuest [20] for selecting paths and for inferring all-path information from measured values. In the following, we first briefly describe the NetQuest framework and then describe how our approach achieves the above mentioned four tasks.

A. Background: NetQuest

NetQuest [20] is a network measurement framework that applies a Bayesian experimental design to select paths to measure that maximize the amount of information subject to the restriction on the number of paths. NetQuest consists of two components: *design of experiment* and *network inference*. The former phase requires a routing matrix that reflects the topology of the network between all end-hosts and uses that information to select user specified number of paths to monitor actively. The paths are selected such that the accuracy of the inferred paths as estimated by the second phase is maximized.

The problem of selecting a subset of a fixed size from the set of all paths that minimizes the error in inference is NP-Complete. NetQuest employs a greedy heuristic for this phase—start with an empty set S for selected paths and keep adding a path that minimizes certain Bayesian *criteria* until the size of S reaches the user specified size (please refer to [20] for more details of the algorithm). For the the total number of paths to measure, we set NetQuest to choose the rank number of the routing matrix. In [19] and [27], Chen *et al.* show that with the measured values of paths corresponding to the independent rows of the routing matrix of a topology, end-to-end performance on every path can be *exactly* reconstructed.

B. Resource Requirement Characterization

We currently focus on the LossDelay sensor and target CPU usage as the resource for this paper. So, to characterize the amount of CPU used by an instance running LossDelay tool, we execute the tool for a long time (≈ 20 minutes) and determine the average CPU used by the tool using UNIX *time* command: (user+sys)/real time. Since the characteristics of machines in a large system can be very different, we perform this task on each node separately. From Figure 1(a), we observe that the CPU usage of the LossDelay sensor grows linearly with the number of instances of the tool running on the machine. We leverage this result to extrapolate the resource usage of one instance to several concurrently running instances.

C. Monitoring Resources

On each machine, we continuously monitor the fraction of CPU used by other processes and determine the number of LossDelay measurements we can initiate from that machine while using only a small fraction of the remaining CPU. For PlanetLab testbed where we perform our experiments, we determine the current CPU usage on each machine using

```

1   $S = \emptyset$            // initialize the path set to be empty
2   $C = \infty$           // initialize the current criteria value
3  while  $C > C_{rank}$ 
4    for each path  $(i, j) \in n \times n - S$ 
5      if  $|\{(k, l) \in S : k = i\}| \leq const_i$ 
6        // compute design criterion with path  $(i, j)$ 
7         $criteria[(i, j)] = \phi(S \cup \{(i, j)\})$ 
8      else
9        // make the path unselectable
10        $criteria[(i, j)] = \infty$ 
11     end if
12   end for
13   // select the path that minimizes the design criterion
14    $p = \arg \min_{(i, j)} criteria[(i, j)]$ 
15    $S = S \cup \{p\}$ 
16    $C = criteria[p]$ 
17 end for
18 return  $S$ 

```

Fig. 9. Path selection algorithm with greedy heuristic.

CoTop [28] tool which outputs CPU usage of all slices running on a machine. Then based on the free CPU of a machine and the CPU requirements of the LossDelay tool on that machine, we determine the number of instances of the tool that we can run on the machine without violating the resource constraints. We then forward this number to a central location for executing the path selection algorithm.

D. Path Selection

Our path selection algorithm is an extension of NetQuest’s design of experiment phase and it determines the list of destinations for each of the end-hosts. Similar to NetQuest, we employ a greedy heuristic but restrict which paths we can add to our selected subset of paths based on the resource constraints.

In Figure 9, we present our path selection algorithm. Here C_{rank} corresponds to the criteria value achieved in the original NetQuest algorithm after choosing rank number of paths. In the algorithm, $const_k$ denotes the number of measurements that we can perform at node k and ϕ is a function that computes the Bayesian design criterion value for a given subset of paths. Without resource constraints, the upper bound on the number of paths can be the rank of the routing matrix because in an ideal world with no measurement error, adding more measurements does not contribute to inference accuracy while using up more resources. However, with resource constraints, it might be impossible to select the smallest path set that result in maximum accuracy gain. Hence, we bound the maximum number of active measurements in our algorithm using the NetQuest’s Bayesian criteria value for the rank number of paths denoted as C_{rank} .

Similar to NetQuest design of experiment phase, our path selection algorithm requires the topology of the network between the end machines. We use S³ deployment on PlanetLab to perform traceroutes from each node to all other nodes on PlanetLab and construct a network topology graph from that data. Note that the topology of the network changes very slowly with time. Initially, we perform all-pair measurements once in a round-robin fashion from each node. Then we

periodically measure on all paths at a slow pace to update the topology. Techniques such as using the remaining TTL from an ICMP response to detect changes in the length of path [20] can be employed to further reduce the overhead of topology maintenance.

E. Measurement and Inference

Once the subset of paths is selected, we propagate the lists of destinations to the corresponding end-hosts. End hosts perform LossDelay measurements to all destination specified in their list. For inference, we gather the measured data from each node to a central location and infer the loss and delay values for the other paths from the measured values using the NetQuest inference algorithm (refer to [20] for more details on the inference algorithm).

F. Discussion: Per-link resource constraints

In this paper, we focus mainly on the loss rate and latency metrics. The LossDelay sensor that we study for measuring these metrics consumes significant end host resources while imposing only a moderate bandwidth requirements. But tools for measuring available bandwidth such as PathChirp can consume significant portion of a link’s bandwidth if several measurements occur on the link. In addition to per-host resource constraints considered in our algorithm 9, we can constrain resources at each link with simple extensions. Instead of considering only end-host constraints in line 4 of Algorithm 9, we can deem a path to be unselectable if any link of the path has already many measurements scheduled on it. While monitoring end-host resources is easy, monitoring per-link free bandwidth is a hard problem. We plan to instead measure capacity of a link and select a subset of paths such that we use only a small fraction of the capacity on each link.

IV. EVALUATION

We evaluate our approach using simulation experiments and via real deployment on PlanetLab. We compare the performance of our resource-aware path selection algorithm to original NetQuest’s resource-oblivious scheduling. The main metric of comparison is the accuracy of inference using the selected paths. The resource-oblivious algorithm chooses a set of paths equal to the rank r of the routing matrix that best represents the topology. But the selection may incur a large number of concurrent measurements on some nodes because this algorithm does not consider each node’s resource availability. In our experiments, if the resource-oblivious algorithm chooses r number of measurements for a node where resource constraint allows only c ($< r$) paths, then we consider only c best paths of the resource-oblivious algorithm. Note that this leads to fewer measured values for the resource-oblivious algorithm to reconstruct the information about other paths while our resource-aware algorithm leverages the constraint information to measure more paths and attain better accuracy in inference. We present the results from our experimentation in the following two sections.

	30 paths	10 paths	5 paths
Resource-oblivious	878	667	425
Resource-aware	1124	938	471

TABLE I
NUMBER OF TOTAL PATHS MEASURED UNDER DIFFERENT PATH CONSTRAINTS BY RESOURCE-OBVIOUS AND RESOURCE-AWARE ALGORITHMS.

A. Simulation Results

In the simulation experiments, we use a dataset gathered using S³ deployment on PlanetLab where we perform traceroutes from each node to all other reachable nodes on PlanetLab. Out of all-pair data, we picked 100 nodes spread across the world. Since traceroutes provide us the latency information, we use only latency inference in these experiments.

In Figure 10, we compare the distribution of paths selected in our path selection algorithm with different resource constraints (number of measurements that we can run at each node). The paths chosen by resource-oblivious algorithm are same as our algorithm with no constraints (shown in the graph as “No const.” curve). We sort the nodes according to the number of outgoing paths from a node and plot with sorted nodes on x-axis in the non-ascending order of the outgoing paths. Note that the resource-oblivious setting has a static selection where it schedules more than 50 measurements on few nodes not considering the constraints on those machines.

In Figure 11, we compare the other aspect of path selection—inference error induced due to various path selections. For each path, we compute the error as the absolute difference between measured and the inferred value divided by the measured value. Then for each experiment, we compute the average of these errors for all paths and denote it as Mean Absolute Error (MAE). As mentioned at the starting of this section, for nodes where resource-oblivious chooses more paths than the node can support, we remove the extra paths. In Table I, we present the total number of path measurements used in each of the algorithms. Observe that for each constraint setting, the resource-aware algorithm uses more measurement data than the resource-oblivious case. For all constraints, our resource-aware algorithm achieves better accuracy than the resource-oblivious setting as we leverage the knowledge of resource constraints to schedule more paths among the end-hosts. The constraint of 5 paths per node is enough restrictive that our algorithm also suffers from the increased inaccuracy but still smaller in comparison to the resource-oblivious algorithm.

B. Real Deployment

In these sets of experiments, we use the topology information gathered by the S³ infrastructure and we actively monitor the CPU load on the machines and perform the path selection on the basis of the remaining CPU. As an initial step, we performed the tool CPU requirement characterization for each machine. We choose the same set of nodes as in the simulation data. When a node is not up, we assume that the free CPU on

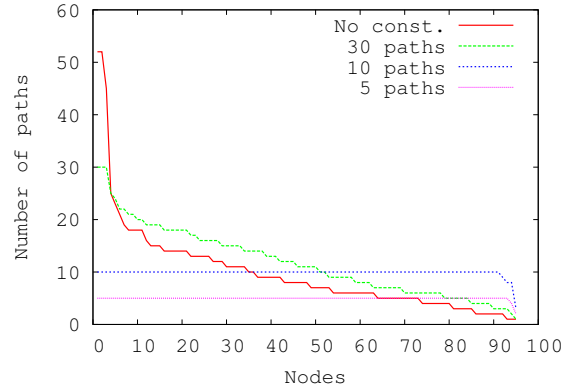


Fig. 10. Comparison of different path count constraints on path distributions in simulation experiments with our resource-aware algorithm.

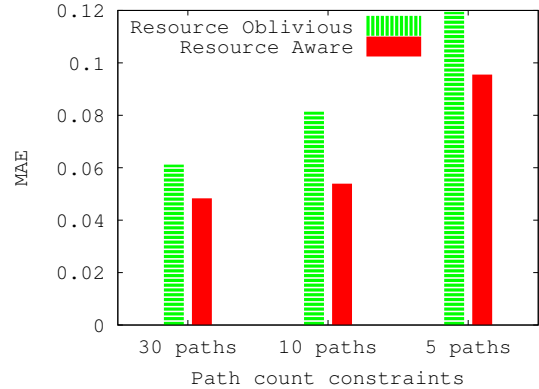


Fig. 11. Comparison of inference error between our resource-aware algorithm and resource-oblivious algorithm with different path count constraints in simulation experiments.

that machine to be 0. For these experiments, we experiment with different resource constraints on CPU: 2%, 1%, 0.5%, and 0.1% of the available CPU.

In Figure 12, we present the constraint on each node (number of outgoing paths) as decided by our resource monitoring process, actual number of paths as selected by our path selection algorithm, and the number of paths chosen by the resource-oblivious algorithm. We sort the nodes in a non-ascending order according to the constraints on the nodes. Observe that without information on resource constraints, resource-oblivious algorithm can schedule a large number of measurements on a loaded machine which is particularly evident when we impose tighter resource constraints of 0.1% in (a) and 0.5% in (b). In Figure 13, we plot the inference accuracy for each resource constraint case comparing our path selection against resource-oblivious setting. By selecting more paths that do not violate resource constraints, our resource-aware algorithm achieves better accuracy than the resource-oblivious algorithm even under heavy constraints.

In Figure 14, we focus on one node and present how our system adapts to the monitored load on the system. We show the target CPU as decided according to our resource constraints, which is 1% of free CPU in this case. We also show the CPU used by our measurements, and CPU used

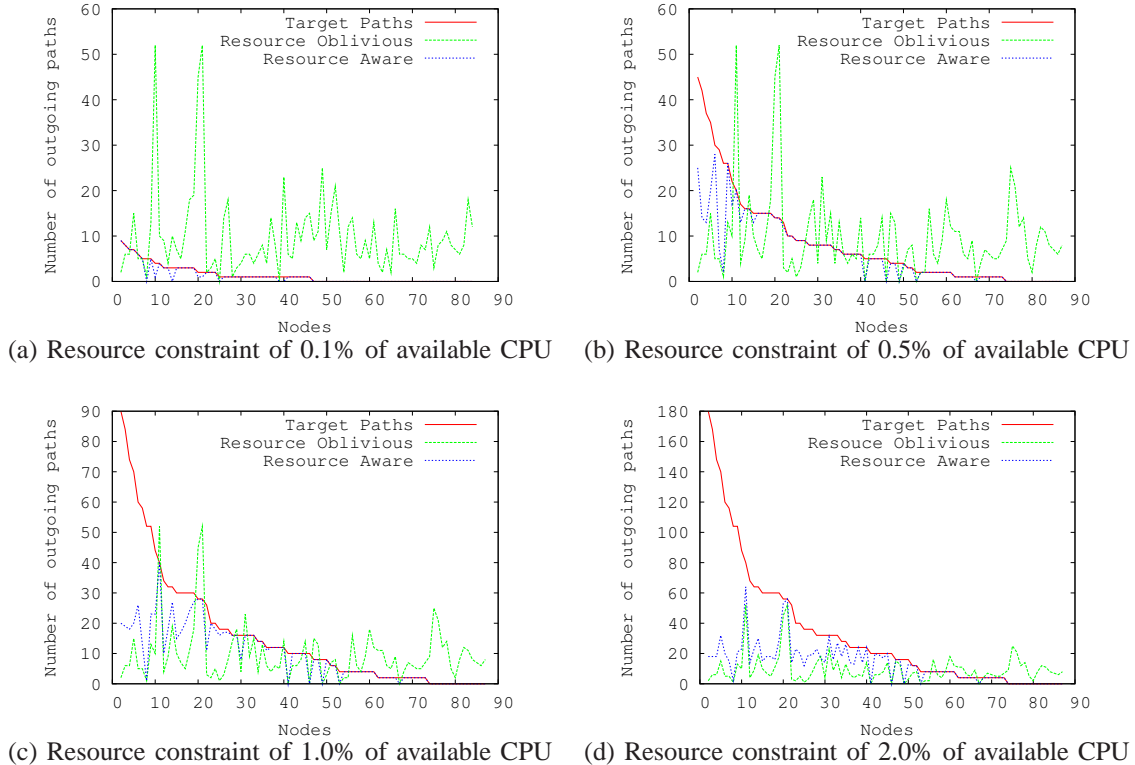


Fig. 12. Path distributions of our resource-aware and resource-oblivious path selection algorithms for different resource constraint experiments on PlanetLab. We also show target paths at each node for each resource constraint case.

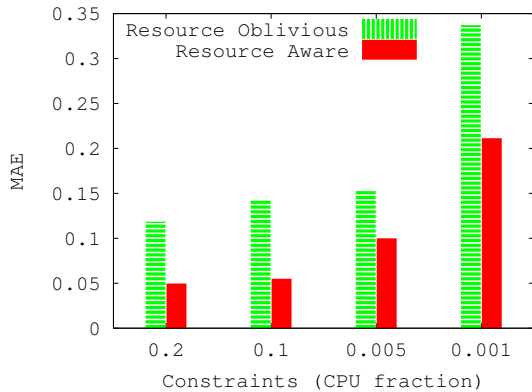


Fig. 13. Comparison of inference accuracy between resource-aware and resource-oblivious algorithms in PlanetLab experiments.

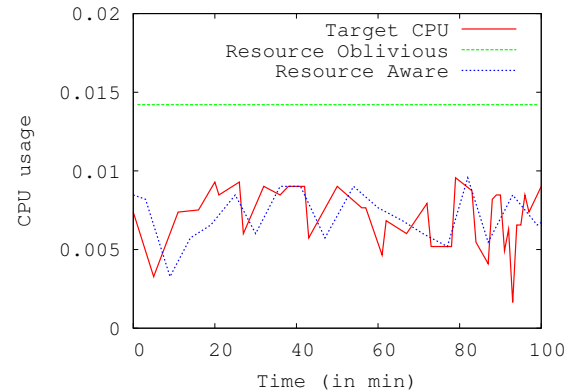


Fig. 14. Adaptiveness of our resource-aware algorithm with changing target CPU on a node in PlanetLab experiment. We also show the CPU required for resource-oblivious algorithm’s selected measurements for this node.

by resource-oblivious algorithm based measurements. Though our system is not entirely optimized yet, it can react to the changes in the CPU load in few minutes. Also, remember that we impose a very strict resource constraint of using only a small fraction of free CPU so that we can easily adapt to any large fluctuations in the CPU usage.

Overall, our simulation results show that our algorithm can leverage the knowledge of resource constraints to select paths that increase the accuracy of the inference results. Our real deployment experiments demonstrate that our system can adapt to the changes in the resources available for measurements.

V. RELATED WORK

The scalable inferencing (e.g., [19]) of network properties is still in research stages. The latency metric has received the most attention, followed by the loss metric. IDMaps [12], King [13], and M-coop [18] answer latency estimation queries for any source and destination nodes by composing other measured data and they need infrastructure support. Landmark based approaches such as Netvigitor [17], Landmark ordering [16], GNP [14], and Lighthouse [15] use landmark nodes for network proximity estimation. Much of the inferencing

research focus on how to obtain $O(n^2)$ results with only a linear number of measurements. But we are exploring the similar problem from the perspective of end-node resource constraints. For example, in landmarks based approach, every node contacts each landmark node irrespective of the load on the landmark machines.

Available bandwidth inferencing is a much harder problem as the bandwidth is not an additive metric. But recent research by Hu et al. [21] and Malli et al. [22] present techniques with promising results. Malli et al. [22] use an approach similar to landmark based latency inferencing algorithms where each end-host performs bandwidth and delay measurements to all landmark nodes and choose the landmark with the closest delay to estimate the available bandwidth. Hu et al [21] claim that the bottleneck links are mostly near the edges and hence only measure the available bandwidth on the links near the end hosts to estimate the bandwidth of all paths.

VI. CONCLUSION AND FUTURE WORK

In this paper, we motivate and present our goal to real-time end-to-end monitoring in large distributed systems while ensuring that the measurement overhead is only a small fraction of free node and network resources. We have presented our current system that leverages and extends NetQuest to monitor loss and delay metrics in large networks while adapting the measurement load to the observed load at the end hosts.

Our current work just scratches the surface of hard problems involved in realizing our full goal — (i) Centralized vs Distributed: Currently, our system uses a central server to collect resource statistics at the end hosts, selects paths, and sends back that information to the end nodes resulting in a cycle of about couple of minutes. We are exploring for distributed techniques for path selection to remove this central bottleneck, (ii) Other inference algorithms: we explored only one inference algorithm of NetQuest in this work. Further work needs to be done to leverage other algorithms such as Landmark based inference algorithms (e.g., GNP and Netvigator), both centralized and distributed, and (iii) Available Bandwidth: This is a much harder metric to infer than additive metrics such as latency and loss. We are exploring techniques from recent heuristics for bandwidth inference [21], [22] and also exploring to adopt the NetQuest inference algorithm to bandwidth metric.

ACKNOWLEDGMENT

The authors would like to thank the invaluable feedback on this research from Lili Qiu, Yin Zhang, Sujata Banerjee, Puneet Sharma, and Sung-Ju Lee.

This work is supported in part by a DARPA Contract (N66001-05-9-8904).

REFERENCES

- [1] L. Wang, K. Park, R. Pang, V. S. Pai, and L. Peterson, "Reliability and security in the codeen content distribution network," in *USENIX*, 2004.
- [2] <http://www.bittorrent.com>.
- [3] <http://www.kazaa.com>.
- [4] "HP Video Collaboration," <http://www.hp.com/halo/>.
- [5] "CISCO Telepresence," <http://www.cisco.com/telepresence>.
- [6] "Monitoring and Managing Avaya IP Telephony Traffic," <http://www.avaya-apac.com/downloads/ipt/lb2554dev.pdf>.
- [7] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient Overlay Networks," in *18th ACM SOSP*, 2001.
- [8] S. Rhea, B. Godfrey, B. Karp, J. Kubiawicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: A Public DHT Service and Its Uses," in *ACM SIGCOMM 2005*.
- [9] P. Yalagandula, P. Sharma, S. Banerjee, S.-J. Lee, and S. Basu, "S³: A Scalable Sensing Service for Monitoring Large Networked Systems," in *Proceedings of ACM SIGCOMM Workshop on Internet Network Measurement*, Pisa, Italy, Sep 2006.
- [10] <http://networking.hpl.hp.com/s-cube/PL>.
- [11] "Planetlab," <http://www.planet-lab.org>.
- [12] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "IDMaps: A global internet host distance estimation service," *IEEE/ACM Transactions on Networking*, vol. 9, no. 5, pp. 525–540, October 2001.
- [13] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *Proceedings of the ACM IMW 2002*, pp. 5–18.
- [14] T. S. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proceedings of the IEEE INFOCOM 2002*.
- [15] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, "Lighthouses for scalable distributed location," in *Proceedings of the IPTPS 2003*.
- [16] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proceedings of the IEEE INFOCOM 2002*, June 2002.
- [17] P. Sharma, Z. Xu, S. Banerjee, and S.-J. Lee, "Estimating Network Proximity and Latency," *ACM SIGCOMM Computer Communications Review*, July 2006.
- [18] S. Srinivasan and E. Zegura, "M-coop: A scalable infrastructure for network measurement," in *Proceedings of the IEEE WIAPP 2003*.
- [19] Y. Chen, D. Bindel, H. Song, and R. H. Katz, "An algebraic approach to practical and scalable overlay network monitoring," in *Proc. of ACM SIGCOMM*, 2004.
- [20] H. H. Song, L. Qiu, and Y. Zhang, "NetQuest: a flexible framework for large-scale network measurement," in *SIGMETRICS '06/Performance '06*.
- [21] N. Hu and P. Steenkiste, "Exploiting Internet Route Sharing for Large Scale Available Bandwidth Estimation," in *Proc. of ACM/USENIX Internet Measurement Conference (IMC 2005)*.
- [22] M. Malli, C. Barakat, and W. Dabbous, "Landmark-based End-to-End Bandwidth Inference," in *Proc. of IEEE Infocom Student Workshop (extended abstract) 2006*.
- [23] C. Dovrolis, P. Ramanathan, and D. Moore, "Packet Dispersion Techniques and Capacity Estimation," *IEEE/ACM Transactions on Networking*, Dec 2004.
- [24] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrel, "pathChirp: Efficient Available Bandwidth Estimation for Network Paths," in *Passive and Active Measurement Workshop*, 2003.
- [25] <http://www.emulab.net>.
- [26] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Proceedings of the ACM IMC 2003*, Miami, FL, October 2003.
- [27] Y. Chen, D. Bindel, and R. H. Katz, "Tomography based overlay network monitoring," in *Proc. of ACM/USENIX Internet Measurement Conference*, 2003.
- [28] "CoTop: A Slice-Based Top for PlanetLab," <http://codeen.cs.princeton.edu/cotop/>.