Resource Management in an Active Measurement Service

Ethan Blanton, Sonia Fahmy, Sujata Banerjee

***Abstract*— Many network services such as voice, video, and collaborative applications require an informed view of network characteristics for effective operation. Sharing a network measurement service across multiple applications can significantly reduce measurement overhead, and remove the burden of performing network measurements from individual applications. To be effectively shared, a network measurement service must provide a variety of measurements to applications on-demand, including end-to-end available bandwidth, delay, and loss. We propose such a service, with a focus on quantifying and bounding the impact of *active* measurements on the network resources being measured. Resource bounds are necessary for wide-scale deployment, since not all users of the service can be trusted. The service informs applications of how service bounds affect their measurements. We introduce methods to characterize the behavior of active measurements for use in admission control and scheduling decisions. We evaluate our methods in experiments under realistic scenarios on the Emulab testbed.**

I. Introduction

Many modern applications and services need information about network properties for effective operation. For example, the Azureus BitTorrent client uses latency information to select peers from which to transfer data [10]. The Tor anonymous communications service [4] reports that network measurement is critical to meeting the challenge of low-latency, high-bandwidth anonymous routing. End-system Multicast [3] hosts use network measurements to build a multicast content distribution tree. A significant number of experimental overlays running on the PlanetLab infrastructure utilize network measurement in their operation, leading the PlanetLab team to suggest a "Routing Underlay" service [13] to serve their needs.

In light of this trend, and with an eye toward next-generation applications, we propose a network measurement service to be used by applications and users desiring a deeper view of network characteristics. This service focuses on bounding the network impact of *active* measurements, while providing the applications which use it with explicit information about the effects of these bounds on the measurements they request. We believe that this is critical to seeing such a measurement service deployed on a large scale.

Active measurements consume network resources. For this reason, it is important that an on-demand measurement service has bounded impact on the network it is measuring. Additionally, a measurement service which is available to untrusted parties but does not limit the impact of its measurements could be readily used as a tool to launch denial of service attacks. We therefore conclude that an open network measurement service must respect administratively-defined resource bounds on the measurement overhead. We define a *load invariant* to ensure resource consumption of measurement tools does not exceed a given budget, and we show how this invariant can be preserved as new active measurement requests arrive.

We have implemented a prototype measurement infrastructure, and used it to evaluate methods for estimating the impact of measurement tools, and the effects of these estimates on the scheduling of active measurements. Our evaluation is conducted on the Emulab testbed [16] using real measurement tools in realistic scenarios.

Ethan Blanton and Sonia Fahmy are with the Department of Computer Science, Purdue University. E-mail: {eblanton, fahmy}@cs.purdue.edu. Sujata Banerjee is with Hewlett-Packard Labs. E-mail: sujata.banerjee@hp.com.

This research is sponsored in part by a gift from Hewlett-Packard, an Intel fellowship, and NSF CAREER grant 0238294. The authors would also like to thank Puneet Sharma and Praveen Yalagandula (HP Labs) and Ahmed Amin (Purdue University) for several helpful discussions on this work.

II. Related Work

The Scalable Sensing Service (S^3) [17] is a measurement infrastructure for large-scale distributed and federated systems. It exports data from a number of measurement tools which are run on the PlanetLab infrastructure. S^3 provides a fixed set of measurements run on a firm schedule, rather than invoking measurements on-demand for any arbitrary application. iPlane [11] also utilizes a fixed schedule.

ScriptRoute [15] provides a programmatic measurement interface to applications. Users of the ScriptRoute measurement mesh submit measurement application scripts to the mesh, which are then executed by ScriptRoute nodes with policies that ensure that measurement impact is bounded. This bound, however, is enforced by external filters, and is not predictable to the application. The application can determine *a posteriori* if its measurement traffic was affected by the filters, but it cannot plan or correct for filter actions.

Calyam *et al.* [2] present a scheduling algorithm to minimize interference between individually scheduled active measurement requests. Measurements which would invalidate the results of other simultaneous measurements are rescheduled or rejected, improving predictability from the application point of view. The paper additionally presents the concept of a Measurement Level Agreement for the regulation of inter-domain network measurement traffic. While their analysis concentrates on reducing conflicts between active measurements, this paper focuses on methods for ensuring that active measurement schedules obey their resource bounds. In short, our work can be integrated into all three systems discussed above.

III. System Architecture

We propose an *open* infrastructure in which users do not require strong authentication or *a priori* enrollment. *Transparency* is a key design principle in our infrastructure. By transparency, we mean that admission control decisions, known interference among measurements, stale measurement information, and other artifacts are communicated to users so that they may be accounted for. We also design to be *dynamic*, by allowing any user to schedule measurements on-demand. *Simplicity* of our user interface allows applications to query network status without having to understand the measurement tools or their operation.

Measurement requests can be made to *any node* that is part of the measurement infrastructure. Clients can also proxy through a special node which communicates with the infrastructure on behalf of the user, providing a standard interface, such as a web server with request forms and output pages. The organization of measurement infrastructure nodes is decentralized, so that any party having an available node to donate may join the infrastructure to provide measurement services. Measurement nodes locate each other by way of a distributed naming system, such as a distributed hash table (DHT) structure, allowing nodes to join and leave at arbitrary times.

When a new measurement request is issued, its resource requirements are drawn from a measurement tool database, and its end points are queried to determine whether they can support the additional load. This load can include, for example, the inbound and outbound access link bandwidth requirements, CPU load, and memory requirements. If all end points can support this load, the measurement is admitted and scheduled; otherwise, it is rejected and the requesting process is notified. We utilize TCP connections for sending measurement requests and their responses. Successful measurement results may be cached for a period of time and returned in case of similar future requests.

The two end points of a measurement tool invocation are infrastructure nodes located in close proximity to end users, such that the measurements between them adequately approximate measurements between end users. Alternatively, the infrastructure nodes can process the measurement results before returning them to the users, to account for the network conditions between the infrastructure nodes and user machines. Such processing can be informed by measurements performed between the measurement host and the requesting end host. This allows the infrastructure to scale well to the Internet scale, in a similar vein as the Domain Name System (DNS).

IV. Admitting Active Measurements

To determine measurement schedules, we utilize the algorithm for scheduling periodic measurements described in [1], which turns out to be similar to that of Calyam *et al.* [2]. Our algorithm uses estimations of measurement tool behavior to build a schedule which serves as many active measurements as can be admitted without exceeding administratively defined resource bounds.[1] A key component to the effectiveness of such an algorithm is the method used to estimate tool behaviors. In this section, we investigate measurement tool requirements and methods for admission control.

A. Assumptions

The admission control and scheduling algorithms make the following assumptions: (1) While end hosts need not be trusted, measurement infrastructure nodes are trusted. (2) Time is encoded as an absolute time in UTC, and clock drift among nodes is small. (3) Scheduled measurement events cannot be preempted. (4) We only consider load bounds on the *sender* and *receiver* nodes of active measurement traffic, and their access links to the Internet. An access link is typically one of the first 4 hops to/from that node. We believe this to be a reasonable assumption because, in today's Internet, access links are typically the bottleneck [6].[2]

B. Measurement Tool Characterization

There is a plethora of tools to measure several aspects of network performance, such as latency, jitter, path bottleneck bandwidth, path available bandwidth, packet loss, and packet reordering. Each measurement tool in our system is associated with a cost vector which approximates its resource requirements over the duration of a single invocation of the tool at both the sender and receiver. This includes inbound and outbound bandwidth requirements, as well as other resource requirements like CPU, memory, processes, and open TCP connections. We currently only consider the *inbound and outbound bandwidth* requirements at each node.

Table I summarizes information about the four popular tools we utilize in our prototype. These tools were selected to be a representative set, both in terms of the quantities they measure and the resources they consume. For each tool, we give the primary property it measures as well as broad descriptions of its resource consumption and how long it typically takes to produce an estimate. We select only four tools for two reasons. First, many measurement tools produced by the research community are not currently suitable for usage in a measurement infrastructure, due to invocation or resource contention limitations. Second, we are not attempting to quantify every possible measurement tool behavior, because new measurement tools are being produced every day.

[1] Observe that if we know all the requests *a priori*, and have perfect estimates of their bandwidth requirements, then maximizing the number of admitted requests within load bounds at any given time becomes an instance of the NP-complete Knapsack problem.

[2] Techniques such as using BGP "atoms" as in iPlane [11] or using correlation tests on packet delays as in [8], [18] can be employed to identify a shared bottleneck that is not an access link, but some other link in the underlying network.

TABLE I
A TAXONOMY OF MEASUREMENT TOOLS.

Tool	Property	Resources	Duration
`ping` [7]	latency	<1 kbit per probe	1 RTT per probe
Pathrate [5]	bottleneck bandwidth	function of end-to-end bandwidth	function of end-to-end delay; typically ~ 20 min.
pathChirp [14]	available bandwidth	~ 1 Mbps peak	~ 10 min.
Tulip [12]	loss, reordering	~ 20 kbps peak	linear function of end-to-end delay

C. Admission Control Tests

We investigate two approaches for our admission control test to ensure that the load invariants are preserved.[3] Both of these approaches use empirical observation of the operation of measurement tools to estimate their resource requirements and duration of execution. Our first approach simply uses the peak bandwidth of each measurement tool over a fixed interval of time. This peak bandwidth is calculated as the maximum number of bits transferred in any interval of the specified length, divided by the interval length in seconds. Separate figures are kept for inbound and outbound bandwidth for every participating host. Admissible traffic bounds are expressed as a simple inbound and outbound scalar. A measurement is admissible if its bandwidth requirements (drawn from its cost vector in the tool database), when added to the existing measurements scheduled at the same time, do not exceed these admissible traffic bounds.

To capture the high variability in instantaneous bandwidth of many measurement tools, our second approach uses the average bandwidth over a fixed interval of time. The measurement tool execution duration is discretized into windows of a specified interval of time, and the amount of bandwidth utilized during each of these windows is computed separately. This series of values is summed and divided by the number of intervals, yielding an average value. The averages for both inbound and outbound traffic of the two end points are used to characterize the tool.

We explored using logical analysis of the algorithms used by measurement tools as a method of estimating their resource requirements, but found that this was extremely challenging. For example, by examining the Tulip [12] source code, we determined that its running time for loss detection on a lossless network should be roughly the product of two run-time configurable constants: the number of probes sent and the delay between these probes. However, in empirical measurements on the Emulab testbed [16], Tulip required a little more than twice this long to complete its measurements. Further analysis revealed that the target of the probes was only sending response packets to about one third of the total number of probes, causing the sender to pause for an extra period to wait for responses which may be delayed in the network. Tulip was able to disambiguate these missing responses (perhaps stifled due to ICMP rate limiting) from true loss, but their effect on the running time is not reliably predictable from examination of the Tulip mechanisms.

[3]Note that admission control tests from the literature [9] cannot be directly leveraged here due to several reasons. First, the admission control literature does not schedule flows to begin at arbitrary times in the future. Therefore, it can utilize online measurement-based techniques, whereas we cannot. Second, the resource requirements of typical active measurement tools are highly bursty. Third, in contrast to the admission control literature which focused on loss and delay guarantees for individual flows, our requirements are defined as aggregate resource constraints on nodes and links. Finally, we can exploit end system scheduling flexibility with respect to the times of invocation of measurement tools.

D. Eliminating Redundancy

If a measurement service handles measurement requests for a large number of applications and users, the possibility of redundant measurements arises. Consider a user that requests loss measurement between a pair of hosts every 10 minutes, and another user that requests loss measurement between the same pair of hosts every 20 minutes starting at the same time. Clearly, we can easily satisfy the lower frequency requester from the information returned to the higher frequency requester.

In practice, the start times or periods of requests may not be exactly identical. The more likely scenario is that they are close to each other. Therefore, we allow each measurement request to include a parameter *pm* (for plus or minus), which denotes that a measurement event can be scheduled with some flexibility. Specifically, a measurement request may be served by any invocation occurring within *pm* seconds of its requested time. This allows us to conflate similar measurements even when their start times are not identical or their periods are not exact multiples of each other. For example, a loss request with a 10-minute period and another with a 15-minute period can be conflated if $pm = 5$ minutes for the latter request.

V. PROTOTYPE AND EXPERIMENTAL EVALUATION

We have implemented a prototype of the admission control and scheduling mechanisms summarized in Section IV (more details can be found in [1]). We give results from a representative set of experiments with our prototype on the Emulab testbed [16]. We select Emulab experiments (rather than Internet experiments, e.g., using PlanetLab) for our initial prototype for three primary reasons. First, we need the results with different mechanisms to be comparable so we can understand which ones work best in depth. Second, we need to easily compare our results with a known "ground truth." Finally, we would like to systematically explore the impact of varying *underlying network* properties, such as link propagation delays and cross traffic, which cannot be controlled on the Internet or PlanetLab.

Our experiments aim to: (1) quantify the benefit of applying admission control; (2) compare different admission control tests and timescales; (3) show that simple estimates of resource requirements of tools such as those described in Section IV-C adequately capture the behavior of network measurement tools; and (4) demonstrate that our admission control tests

are sufficiently robust to reasonable variations in network behavior, such as latency and cross-traffic.

A. Experimental Scenario

Fig. 1 gives the topology we use. The link connecting the core ring to the leaf subnetworks is what we consider to be the bottleneck access link. We choose values to represent a modem user, DSL user, cable modem user, T1 link, and T3 link. Each edge router has a cluster of three hosts connected to it, similar to those depicted on the DSL and T3 networks in the diagram. Unless otherwise specified, the links connecting these hosts are 100 Mbps Ethernet with an artificial delay of 2 ms, 2 ms, and 3 ms on the DSL cluster, and 4 ms–15 ms for the remaining hosts, increasing by 1 ms per host, clockwise around the ring as shown. The measurement infrastructure node in each cluster is the "center" of these values. For convenience, we number the hosts as *host 1* through *host 15*, again clockwise starting with the first DSL non-measurement host. The queue depth is 50 for all queues, and all queues are drop-tail.

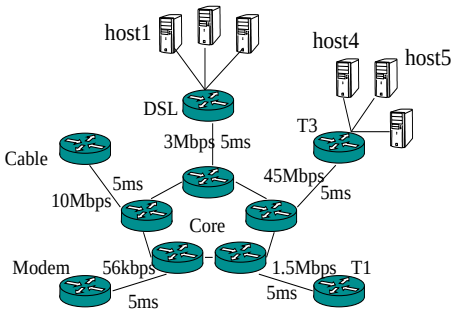

Fig. 1. Experimental topology on Emulab. The links in the core network are unmodified 100 Mbps Ethernet links.

Unless otherwise specified, we configure the hosts to use either peak or average bandwidth for admission, and we submit a fixed set of randomly-selected measurement requests to the system. The tool to be run, source host, and destination host of each request are uniformly and randomly selected from the four tools and five measurement hosts described above; measurements which would use the same host for source and destination are discarded. The start time, repetition period, and repetition count for each request are also randomly selected, such that each request starts and ends within a 45-minute window. Each measurement node sets its inbound and outbound bandwidth bounds to 20% of the bandwidth of its access link over the entire experimental duration. We use 1 second for the duration of the time interval for the peak and average admission control methods and set *pm* to zero, unless otherwise specified.

To estimate measurement tool resource requirements for these experiments, we ran each tool used in these experiments between all pairs of measurement hosts on our experimental network. We found that the behavior of some measurement tools (notably Pathrate) is sensitive to bottleneck link bandwidth (see Table I). Therefore, we use the appropriate estimates from this data set for each measurement host, to accommodate the large discrepancies in link bandwidth in our experimental topology (specifically, the modem link versus the T3 link). In practice, each infrastructure node would select from a *small set* of resource requirement estimates based on the range in which its access bandwidth falls.

TABLE II
AVERAGE RESULTS OF TEN 25-MEASUREMENT WORKLOADS. BANDWIDTH VALUES ARE IN KBPS.

Host	Requested	Failed	bw_{in}	bw_{out}	V_{in}	V_{out}
No Admission Control						
T3	13.2	0	441	359	0	0
DSL	11.7	0	199	451	345	838
Modem	11.8	0	15	20	602	1151
Cable	13.3	0	446	209	0	0
Average Admission Control						
T3	13.2	3	278	295	0	0
DSL	11.7	4	129	247	130	415
Modem	11.8	6.6	4	4	223	224
Cable	13.3	3.6	311	154	0	0
Peak Admission Control						
T3	13.2	5.1	113	211	0	0
DSL	11.7	5.7	31	57	0	0
Modem	11.8	7.8	3	3	78	56
Cable	13.3	5	217	91	0	0

B. The Case for Admission Control

We first compare the peak and average methods with the case when no admission control is employed. The measurement request workload is created by taking 25 random periodic measurement requests and scheduling them across the DSL, modem, cable, and T3 measurement hosts. The same set of workloads was repeated for each admission control method. Results are the average of 10 such workloads. Each experiment runs for approximately 45 minutes (2700 seconds). We measure the traffic created by measurement tools under this scenario, and compute violations of the specified bounds and the measurements admitted.

Table II gives the results. All bandwidth values (bw_{in} and bw_{out}) are in kbps, and represent the average bandwidth utilized at a measurement node over the course of the entire experiment. The "Requested" and "Failed" columns represent the absolute number of measurement requests involving a given host, and the number of those which were not admitted, respectively. The V_{in} and V_{out} columns represent the number of seconds when the inbound or outbound load bounds, respectively, were violated.

We note that the modem host suffers a significantly larger number of resource bound violations than the other hosts. This is due to the extreme constraint imposed by the modem's bandwidth capability, causing even minor estimation errors to lead to oversubscription of bandwidth.

As expected, the average bandwidth usage and number of load bound violations increases steadily with the workload for the scenarios using no admission control [1]. With the average and peak admission control methods, however, the average bandwidth usage peaks very close to this 25-measurement workload, and violations likewise level off. The average bandwidth estimation scheme, by its very nature, allows some violations to occur. However, these violations represent only

a relatively small portion of the 2700 second duration of the experiment. Hence, the average method trades off strict adherence to the load invariant, for satisfying additional measurement requests.

C. Estimation Accuracy for a Typical Workload

The goal of our next series of experiments is to investigate the accuracy of using the peak and average bandwidth estimations as discussed in Section IV-C in more depth, using a typical request workload. To formulate a typical measurement request workload, we create three "roles" for the end points. The hosts are divided among those participating in an end-system multicast video stream, online gaming, and a system administrator measuring end-to-end capacity. Within each group, the selected measurements are requested in a complete mesh. The end-system multicast hosts require latency information every 10 seconds and available bandwidth information every 15 minutes, provided by `ping` and pathChirp, respectively. These hosts are located on the T3, T1, and cable networks. The online gamers are on the modem and T1 networks. They require latency information (via `ping`) every 10 seconds, and loss information (using Tulip) every 5 minutes. Network capacity is measured using Pathrate, which is invoked once between the T3 and DSL networks. All requests are scheduled starting at time 0 and complete all repetitions within 45 minutes.

Fig. 2 depicts the measured outbound traffic on the measurement host behind the T3 line, named "host5" in Fig. 1. The plot also shows our estimates of the traffic on this node. For the peak scheme, we find that the estimated bandwidth usage is much higher than the actual bandwidth seen. The peak scheme pessimistically assumes that the moment of maximum utilization of each overlapping measurement will occur at the same time, yielding a peak bandwidth utilization which seldom occurs in actual usage. Based on these results, it appears that the average bandwidth estimate is better suited for a measurement service that schedules events at arbitrary times in the future. This is consistent with the results summarized in Table II.

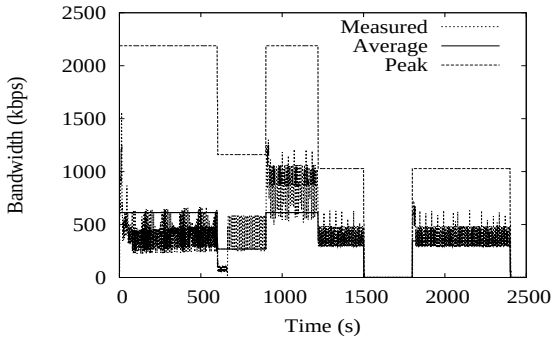

Fig. 2. Outbound traffic seen emanating from host5, as well as the estimates using the average and peak bandwidth schemes.

D. The Impact of Changing Network Conditions

Our next series of experiments validate that measurement admission control is viable even under some degree of changing network conditions.

Cross Traffic. Fig. 3 repeats the same experiment with results depicted in Fig. 2, but with the difference that we inject cross traffic. We use the average method for the estimate shown in the plot. Three types of cross traffic were generated, sourced and sinked at the various non-measurement hosts. These types were: HTTP sessions generated by WebStone 2.5, back-to-back bulk TCP transfers of 2048 writes of length 8192 (sourced and sinked with the "ttcp" tool), and variable bit rate "Ogg Vorbis" streams of average bandwidth ranging between 120 and 125 kbps. Each WebStone session was configured to behave as a single browsing client, fetching files ranging in size from 0.5 kB to 5 MB. The specific configuration of cross traffic can be found in [1].

We find little difference between the measured traffic in Fig. 3 and 2. The largest discrepancy between the measured and estimated values in Fig. 3 is during the final burst period of Pathrate: the burst is delayed from about time 650 seconds to around 800-900 seconds, and is shorter in duration. Overall, we find that the admission control methods are robust to changes in cross traffic.

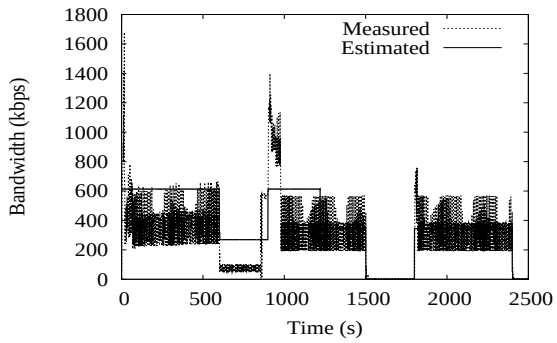

Fig. 3. Outbound traffic seen on host5 for the same measurement set as Fig. 2, but with cross traffic in the network.

End-to-end Delay. To quantify the effects of end-to-end latency on admission control, we conduct a series of experiments with 100 randomly-selected requests, using the average estimator. We vary the delay on each of the "arms" of the star topology (the link between the central ring and the router heading each cluster of hosts) in 5 ms increments from 5 ms to 100 ms, for total end-to-end one-way delays of up to 200+ ms. We investigated correlations between these expanding delays and changes in load bound violations, but found no clear impact.

E. Estimation Timescales

We now evaluate the impact of the timescale over which measurement tools are characterized, and admission tests are conducted. We consider both the average and peak bandwidth admission methods, where the bandwidth is calculated over intervals of 100 ms, 1, 2, and 4 seconds. Table III shows the percentage of accepted measurements. We again utilize a workload of 25 randomly-generated requests in this set of experiments, with the results being the average of 10 such workloads. The experiment setup and workloads correspond to

TABLE III

REQUEST SUCCESS RATES WITH VARYING TIMESCALES.

Timescale	Average	Peak
100 ms	66%	35.2%
1 s	64.4%	47.2%
2 s	66%	49.6%
4 s	66%	56%

TABLE IV

THE EFFECTS OF $pm > 0$ ON MEASUREMENT SCHEDULING. RESULTS SHOWN ARE THE AVERAGE OF TEN EXPERIMENTS.

Method	*pm*	Requested	Invoked	% Saved
Average	0	2991	2831	5.4%
	max	3068	2485	19.1%
Peak	0	3118	2961	5.1%
	max	3121	2513	19.5%

those used for the experiments in Table II. The table illustrates that the average admission control method, as expected, is robust to the timescale over which estimations are made. Load bound violations are also consistent, with variations between estimation timescales being smaller on average than variations between different experiments due to their random workloads. For example, for the peak admission control method, the coefficient of variation between timescales is 0.16, while the coefficient of variation between the random workloads for the 1 second timescale is 0.44. The peak method varies in its acceptance rate, with the shorter intervals admitting fewer requests. Violations under the peak method vary correspondingly, with longer intervals exhibiting more violations. The difference among the two methods can be attributed to the highly bursty nature of measurement tool traffic. The average bandwidth admission method admits a larger number of expensive requests (Pathrate) early on. In contrast, the peak bandwidth admission method "fills in" gaps in its schedule with less expensive requests (Tulip).

F. Eliminating Redundancy

Finally, we evaluate the potential savings of measurement tool invocations when $pm > 0$. Using the measurement request workload from Table II, we conduct experiments with $pm = 0$ and with the maximum *pm* for each individual measurement request.[4] Table IV summarizes the results. The "Requested" column indicates the number of individual measurement tool invocations successfully admitted by the system, and the "Invoked" column represents the number of actual invocations executed. The percentage of invocations saved out of the number requested is listed in the "% Saved" column. As this table illustrates, with a sufficiently dense set of measurement requests, a significant number of measurement invocations can be saved by allowing flexibility in scheduling. This allows for a moderate number of additional measurement requests to be served.

[4]Note that any $pm \geq \frac{1}{2}\times$ the period of a measurement request allows two measurement requests to be served from a single measurement tool invocation.

VI. CONCLUSIONS

We have proposed an open active measurement service, and explored methods for scheduling measurements based on resource consumption. We find that the characterization of measurement tool resource requirements is very important, and that due to the extremely bursty nature of measurement traffic, peak bandwidth utilization is an over-conservative metric. We have demonstrated that average bandwidth utilization is a more accurate metric, but that even this may require refinement in some cases, to account for greatly varying utilization during different phases of execution of tools like Pathrate. A characterization scheme which captures the essence of this burstiness in a more explicit manner may allow better admission decisions to be made, and is the subject of our future work.

REFERENCES

[1] E. Blanton, S. Fahmy, and S. Banerjee. A framework for an on-demand measurement service. Technical report, Purdue University, 2008. http://www.cs.purdue.edu/homes/fahmy/reports/measurement.pdf.

[2] P. Calyam, C.-G. Lee, E. Ekici, M. Haffner, and N. Howes. Orchestration of network-wide active measurements for supporting distributed computing applications. *IEEE Transactions on Computers*, 56(12), Dec 2007.

[3] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proc. of SIGMETRICS*, June 2000.

[4] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proc. of USENIX Security Symposium*, 2004.

[5] C. Dovrolis and P. Ramanathan. Packet dispersion techniques and capacity estimation. *IEEE/ACM Transactions on Networking*, December 2004.

[6] N. Hu and P. Steenkiste. Exploiting Internet route sharing for large scale available bandwidth estimation. In *Proc. of IMC*, 2005.

[7] G. Kessler and S. Shepard. A primer on Internet and TCP/IP tools and utilities. RFC 2151, June 1997.

[8] M. S. Kim, T. Kim, Y. J. Shin, S. S. Lam, and E. J. Powers. A wavelet-based approach to detect shared congestion. In *Proc. of ACM SIGCOMM*, 2004.

[9] E. W. Knightly and N. B. Shroff. Admission control for statistical QoS: Theory and practice. *IEEE Network*, 13, March 1999.

[10] J. Ledlie, P. Pietzuch, M. Mitzenmacher, and M. Seltzer. Network coordinates in the wild. In *Proc. of NSDI*, April 2007.

[11] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An information plane for distributed services. In *Proc. of OSDI*, pages 367–380, November 2006.

[12] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet path diagnosis. In *Proc. of SOSP*, October 2003.

[13] A. Nakao, L. Peterson, and A. Bavier. A routing underlay for overlay networks. In *Proc. of SIGCOMM*, pages 11–18, 2003.

[14] V. J. Ribiero, R. H. Reidi, R. G. Baraniuk, J. Navratil, and L. Cottrell. pathchirp: Effi cient available bandwidth estimation for network paths. In *Proc. of PAM*, 2003.

[15] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A public internet measurement facility. In *Proc. of USITS*, 2002.

[16] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.

[17] P. Yalagandula, P. Sharma, S. Banerjee, S. Basu, and S.-J. Lee. s^3: A scalable sensing service for monitoring large networked systems. In *Proc. of INM*, September 2006.

[18] O. Younis and S. Fahmy. Flowmate: Scalable on-line flow clustering. *IEEE/ACM Transactions on Networking*, 13(2), April 2005.