A Framework for an On-Demand Measurement Service

Ethan Blanton, Sonia Fahmy, Sujata Banerjee*†

Abstract

End-to-end network measurements such as delay, loss, and available bandwidth are essential for optimizing and managing network applications and services. Sharing a network measurement service across multiple applications can significantly reduce measurement overhead, increase accuracy, and remove the burden of performing network measurements from individual applications. Network measurement data is most useful if measurements can be requested on-demand by users at desired *times* and *frequencies*. Clearly, in a federated environment, it is difficult to have the same level of trust in all users making measurement requests. This paper describes a framework to provide network measurement information to *untrusted* users such that *active* measurements can be taken in a safe manner. By safety, we mean that users cannot trigger arbitrarily large measurement probes that exceed a specified budget. We devise efficient admission control and scheduling algorithms for recurring active measurements, such that the specified resource consumption budget of the service is best utilized. We evaluate our approach in experiments under realistic scenarios on the Emulab testbed.

1 Introduction

This paper explores network measurement *as a service*. We consider how to *serve measurement information to untrusted hosts via a large-scale measurement infrastructure in a safe fashion*. We define *network measurement information* as a specific set of measurable network characteristics (e.g., pairwise latency, available end-to-end path bandwidth, traceroute-style path identification) from which requesting hosts may choose a number of characteristics to be measured at user-requested times and frequencies. We consider a *large-scale infrastructure* to be one consisting of dozens to hundreds or more cooperating measurement hosts serving arbitrary Internet or intranet users. Finally, we define a *safe* measurement service as one which honors administratively specified bounds on network bandwidth and measurement host resource consumption.

The importance of a planned measurement infrastructure that can be shared among multiple network users is well-recognized [19, 21, 29, 31]. End-host services which benefit from an informed view of the network are becoming more prevalent. These services include bandwidth- and latency-sensitive multimedia conferencing or streaming systems such as end-system multicast [3], latency- and loss-sensitive VoIP services such as Skype, bandwidth-fair file sharing such as BitTorrent, and server selection protocols, content distribution networks, and distributed storage systems. For example, the Azureus BitTorrent client uses distributed network measurement [17]. Another example is the Tor [5] overlay anonymity service, which cites widespread measurement as one of the challenges to scalability [4]. Distributed and grid computing applications likewise benefit from network health and behavior information [2]. A shared measurement infrastructure allows network operators and managers with limited local resources to obtain a better picture of the state of the Internet or an intranet, or of their own connectivity to the Internet. This can be useful for fault localization and network tuning. *A safe and independent measurement infrastructure can provide measurement data to these applications in a manner which is "friendly" to the network and controlled by its administrators.*

While several research studies have considered mea-

*Ethan Blanton and Sonia Fahmy are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907–2107, USA. E-mail: {eblanton, fahmy}@cs.purdue.edu. Sujata Banerjee is with Hewlett-Packard Labs, 1501 Page Mill Rd., Palo Alto, CA 94304, USA. 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.

surement infrastructures [19, 21, 29, 31], our approach is guided by the observation that the utility of a measurement service is significantly increased by the *extensibility* of the queries it can answer, as well as its graceful handling of heavy measurement request load. Previous work either does not allow on-demand requests, or lets each user independently invoke measurement tools and places static filters on their traffic. In contrast to these two extremes, we design a flexible yet safe service for handling on-demand queries, and devise effective *a priori* admission control and scheduling strategies to ease the burden of obtaining accurate and predictable measurements for applications.

In this paper, we focus on the question of how to best utilize a given budget of network resources that is allocated to the measurement service. We observe that there are trade-offs among (i) the resources consumed by measurement tools (e.g., bandwidth consumed by active probes), (ii) the flexibility of the times at which measurements are required, (iii) the required accuracy of the measurements. These tradeoffs can be exploited to bound the resource consumption of measurement tools, and to reduce interference among them. For example, if the required measurement accuracy is not high, low cost measurement methods can substitute for high cost ones to reduce resource consumption. Another example is using a single measurement invocation to satisfy two nearby users requesting loss measurements to the same server network at different times but with an indication that measurement times are flexible within a certain range. This reduces resource consumption and allows more requests to be admitted.

We define a *load invariant* to ensure resource consumption of measurement tools does not exceed the given budget, and we show how this invariant can be preserved as new active measurement requests are submitted to the system. The careful admission control of measurement requests prevents the use of the measurement infrastructure as a denial of service tool, and allows the administrators of measurement hosts to prioritize the measurements they admit according to local policy. Flexibility in the requested measurement times and required accuracy eliminates redundant measurements that can use an undue share of network resources[1]. Independent handling of network measurement by each interested party (as in today's Internet and many of today's measurement infrastructures, e.g., [29]) causes the measurement load on the network to be on the order of the number of properties measured times the number of processes interested in each property. With a planned measurement infrastructure, we can reduce this to a single probe for each property per pair of infrastructure hosts.

[1] PlanetLab sees over 1 GB/day of outbound `ping` traffic alone [21], due to redundant probing.

Our primary contributions in this work include: (i) Definition of an architecture for a flexible yet safe measurement service; (ii) Algorithms for admission control and scheduling of recurring measurement requests; (iii) Implementation of a prototype system, and (iv) Comparison of simple admission control and scheduling techniques for measurement requests using real experiments on the Emulab testbed.

The remainder of this paper is organized as follows. Section 2 describes the architecture of our network measurement service. Section 3 gives our definitions and assumptions, and discusses characterization of active measurement tools. Sections 4 and 5 give our core algorithms for scheduling and admission control. Section 6 gives results from our experiments. Section 7 summarizes related work. Finally, section 8 includes a discussion of our results and directions for future work.

2 System Architecture

We propose an *open* infrastructure in which users do not require strong authentication or *a priori* enrollment. This spurs our emphasis on honoring administratively-defined load bounds on measurement hosts. A second guiding principle is *transparency*. By transparency, we mean that admission control decisions, known interference among measurements, stale measurement information, and other artifacts can be 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 rather than relying on fixed schedules as in [19, 31], or serving least-common-denominator information. Finally, a primary concern in the user interface is *simplicity*, such that applications can effectively 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 a measurement infrastructure, or to 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, e.g., [24], allowing nodes to join and leave at arbitrary times without need for centralized administrative action.

When a new measurement request is issued, its resource requirements are drawn from a measurement tool database (discussed in section 3.3), and its two 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 of this rejection. 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).

3 Preliminaries

In this section, we define measurement requests and tools more precisely, and give the assumptions we make in this work.

3.1 Definitions

Consider a set of measurement hosts, $h_1, h_2, \ldots, h_n$. Assume that an end-to-end active measurement request r (defined below) is issued to host h_i (sender), which will perform its task by sending a stream of packets (typically referred to as *probes*) to h_j (receiver). Each of h_i and h_j will independently perform an admission control (schedulability) test for request r, and if both end points can admit r independently, then r will be admitted to the system. For each admitted request, the hosts will compute a *measurement event* e. Scheduling e will preserve a specified load invariant that ensures load bounds are maintained.

Definition 1 *A measurement request r is represented as a tuple (P, T_0, T_p, T_c), where P is the property being requested (such as "latency" or "available bandwidth"), T_0 is the start time for the first measurement, T_p is the* period *(i.e., reciprocal of frequency) at which P should be measured (e.g., every 30 minutes; T_p is set to infinity for one time only measurements), and T_c is the measurement repetition count, which can be infinity if the measurement should be periodically taken indefinitely into the future. A measurement will be admitted if all repeating instances of the measurement can be successfully admitted on all hosts involved, without violating any load bounds.*

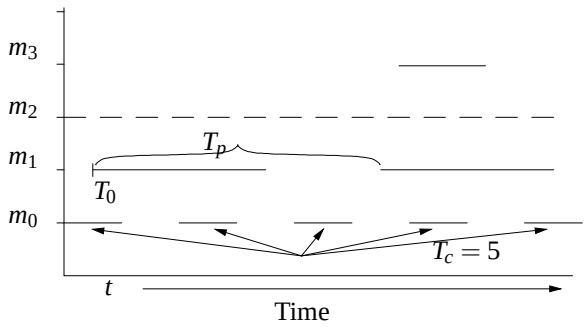

Figure 1: A set of scheduled measurement requests m_0 through m_3. T_0 is the start time of a scheduled measurement event; T_p is its period; and T_c is its repetition count.

A measurement event e is represented as a tuple (M, T_0, T_p, T_c), where M is the name of the measurement tool we wish to invoke (including any relevant configuration information), and T_0, T_p, T_c are the same as in the measurement request. The value of M could be, for example, the standard `ping` utility [14], or a tool such as Pathrate [7], pathChirp [25], or Tulip [20]. Fig. 1 depicts a set of measurement requests, illustrating the T_0, T_p, and T_c parameters.

3.2 Assumptions

We make the following assumptions in our work:

(1) Trust: We assume that a set of approved measurement tools has been installed on the measurement infrastructure machines, which are assumed to be trusted and to be running our software. This is a reasonable assumption, since the infrastructure nodes will be under the administrative control of system administrators, in the same manner as local DNS servers, for example. A new measurement tool must undergo a basic validation and vulnerability assessment process before installation on any infrastructure node. Packet filters can then be installed on the infrastructure node to enforce the expected behavior computed during the validation process. Section 3.3 discusses the set of measurement tools we are utilizing in our prototype.

In contrast to infrastructure nodes, users making measurement requests are *not* assumed to be trusted; they may submit unreasonable requests to attempt to overload the service or launch denial of service attacks.

(2) Load bounds: For the purposes of admission control, 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

Table 1: A taxonomy of measurement tools.

Tool	Property	Secondary	Resources	Duration
`ping` [14]	latency	jitter	$<$ 1 kbit per probe	1 RTT per probe
Pathrate [7]	bottleneck bandwidth	latency	function of end-to-end bandwidth	function of end-to-end delay; typically $\sim$ 20 min.
pathChirp [25]	available bandwidth	latency	$\sim$ 1 Mbps peak	$\sim$ 10 min.
Tulip [20]	loss, reordering	latency	$\sim$ 20 kbps peak	linear function of end-to-end delay

one of the first 4 hops to/from that node. We believe this to be a reasonable assumption because in today's Internet, backbones are mostly underutilized and experience little queuing and loss [10, 11], while the access links (sometimes referred to as the "last mile") are typically the bottleneck [12].[2]

(3) Time representation and clock skew: We encode T_0, the start time of the first measurement in a repeating sequence, as an absolute time in UTC. We do not account for clock skew between the measurement infrastructure nodes participating in a measurement in our current prototype. Our system does not, however, require that absolute system time is tightly synchronized between measurement hosts. Mechanisms to deal with time offsets can be engineered without difficulty, provided that clock drift is not excessive.

(4) No preemption: We assume that scheduled measurement events cannot be preempted. In other words, incoming requests are handled on a first-come first-served basis, without giving any priority to any request over another.

3.3 Measurement Tool Characterization

There is a plethora of tools to measure several aspects of network characteristics, such as latency, jitter, path bottleneck bandwidth, path available bandwidth, packet loss, and packet reordering. For most such aspects, multiple tools provide measurements by differing methods and of varying precision. In addition, some tools measure multiple aspects; in some cases, a tool measures one property as a primary goal, but can also provide (possibly low-precision) estimates of other properties. Hence, we can perform several optimizations. For example, if a measurement request for a property at a high precision cannot be admitted, and a tool that produces a low precision estimate of that property is scheduled along the same path at the same times (or can be scheduled within the load bounds), we can leverage the results from that tool to satisfy the user request, indicating to the user that the precision may be low.

We associate each measurement tool 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 (see [27] for end system resource requirements of measurement tools). We currently only consider the *inbound and outbound bandwidth requirements* at each node.

Table 1 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. 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. Modifying these tools to be used in such an infrastructure is a time-consuming process. Second, we are not attempting to quantify every possible measurement tool behavior, because new measurement tools are being produced every day. We simply select a representative set of tools for evaluation of the framework itself. For each tool, we give the primary and secondary properties it measures, as well as its resource consumption and how long it typically takes to produce an estimate.

Unfortunately, accurately formulating the resource requirements of each tool as a function of the underlying network conditions is challenging. For example, by examining the Tulip [20] 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 [8], Tulip required a little more than twice this long to complete its measure-

[2] Techniques such as using BGP "atoms" as in iPlane [19] or using correlation tests on packet delays as in [15, 32] can be employed to identify a shared bottleneck that is not an access link but some other link in the underlying network.

ments. 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 simply 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. Section 6.6 further discusses such challenges, and experimentally explores the sensitivity of the traffic injected by measurement tools to cross traffic and end-to-end latency.

4 Recurring Measurement Requests

As discussed in Section 3, a measurement request may indicate that a property should be periodically measured a large number of times, or even indefinitely. We therefore devise an algorithm that, given a set of measurement events, divides time into a series of discrete partitions for which *performing the admission control test over a specified duration*, referred to as *hyperperiod*, of each computed partition is equivalent to performing the test on the entire continuous timeline. The hyperperiod represents the least common multiple of the periods of individual overlapping measurements. Sprunt [30] shows why hyperperiods are sufficient for schedulability tests.[3]

Fig. 2 gives our algorithm for computing the set of partitions. In the pseudo-code, "append to output" indicates that a tuple containing a list of events, start time, end time, and hyperperiod is appended to the list of partitions being generated as output. The function $period(e)$ returns the period of event e (i.e., its T_p; if $T_p = \infty$ (one-time-only measurement) we handle this as a special case). Functions $start(e)$ and $finish(e)$ return the start time and finish time of the first tool invocation and last invocation in e, respectively. $first(L)$ is a function returning the first element of L, $lcm(a,b)$ gives the least common multiple of a and b, and $removefirst(L)$ removes the first element of L. The variable L stores the list of events being added to the current partition, *ps* denotes the start time of the partition being computed, *pe* denotes the end time, and *hyper* denotes its hyperperiod duration.

The central intuition in this algorithm is that any long-running measurement event, under the assumptions put forth in Section 3, will be strongly periodic in exactly the period T_p (defined in Section 3.1). Therefore, the cost of any set of concurrent measurements is periodic

[3] Sprunt's work extends Liu *et al.*'s rate-monotonic algorithm [18] to handle aperiodic requests. Algorithms from the real-time scheduling literature are not applicable to our problem, however, due to their differing underlying assumptions such as deadline-based scheduling and preemptability.

let E_s be initialized to the sequence of measurement events e_i scheduled on host h, sorted by $start(e_i)$.
let E_f be initialized to the sequence of measurement events e_i scheduled on host h sorted by $finish(e_i)$
let $ps = pe = 0$.
let $L = \emptyset$.
while $E_s \neq \emptyset$:
    let $pe = min(start(first(E_s)), finish(first(E_f)))$.
    let $hyper = 1$.
    for each event e in L:
        let $hyper = lcm(hyper, period(e))$.
    if $L \neq \emptyset$ append to output $(L, ps, pe, hyper)$.
    let $ps = pe$.
    while $E_s \neq \emptyset$ and $start(first(E_s)) = ps$:
        let $L = L \cup \{first(E_s)\}$.
        let $E_s = removefirst(E_s)$.
    while $finish(first(E_f)) = ps$:
        let $L = L - \{first(E_f)\}$.
        let $E_f = removefirst(E_f)$.
while $E_f \neq \emptyset$:
    let $pe = finish(first(E_f))$.
    append to output $(L, ps, pe, hyper)$.
    let $ps = pe$.
    while $E_f \neq \emptyset$ and $start(first(E_f)) = ps$:
        let $L = L - \{first(E_f)\}$.
        let $E_f = removefirst(E_f)$.

Figure 2: Algorithm for partitioning time into segments with hyperperiods of repeating history.

in the period of the *least common multiple (lcm)* of their respective periods T_p, or their hyperperiod. For example, assume there are three measurement requests, each with their first occurrence at the same time t_s, and their final occurrence at the same time t_f, and T_p values of 5, 10, and 25. For this, the hyperperiod will be lcm(5, 10, 25), or 50. If we perform the admission control test over any period of time of length 50 between t_s and t_f, it is equivalent to having checked the entire time between t_s and t_f. Each partition output by our algorithm is a tuple $(L, t_{start}, t_{end}, hyper)$, where L is the set of events which are scheduled during that partition, t_{start} and t_{end} are the starting and ending times of the partition, and *hyper* is the duration of the hyperperiod within the partition that is sufficient to test for admission control purposes.

The viability of a new measurement can be checked by (i) tentatively adding it to the list of accepted measurements, (ii) computing the set of partitions which are created when this measurement is included by using the algorithm in Fig. 2, and (iii) checking the first hyperperiod within each partition for load invariance compliance,

as discussed in section 5.

4.1 Complexity

If E is a set of events scheduled at a particular host, then the complexity of the algorithm in Fig. 2 is $O(|E| \times lcm_{e \in E}\, period(e))$. To see this, note that the number of partitions to be checked is bounded by $2|E|$, as partition boundaries fall only on the beginning or end of events. Within each partition, the maximum computation required is bounded by the hyperperiod of the events within that partition; this number is upper-bounded by the hyperperiod of all events in the system.

4.2 Evaluation

Consider the scheduling of two concurrent measurements between two hosts over a long period of time: a latency probe once every minute and a loss measurement once every ten minutes for three weeks, with the latency probe being scheduled first. Suppose that the infrastructure chooses to satisfy the latency probe with a simple `ping`, and the loss measurement with Tulip, configured such that `ping` sends one packet for each probe and Tulip sends an approximately 320 Kbps stream for a duration of 300 seconds.

Using the period partitioning algorithm without any further optimization, our infrastructure must examine 600 seconds (ten minutes) of history in order to make an admission decision for the second measurement. Contrast this to the naïve approach of simply checking the entire footprint of each measurement, which would require the computation and testing of three weeks, or 1,814,400 seconds, of traffic. Even examining only those time periods which contained an active measurement would require the computation and testing of more than half of this period, due to the dense nature of the Tulip measurements.

Using a simple simulator, we conducted a comparison between the naïve scheduling algorithm and our period partitioning algorithm for this scenario. On a 2400 MHz Pentium 4 PC, the difference between the run times for the naïve algorithm and period partitioning was more than 145 seconds. The entire simulation run, including simulator setup and the loading of the scenario from file, took less than 800 ms with our period partitioning algorithm. We do not claim that either implementation is optimal; in particular, our choice of an interpreted implementation language (Ruby) is likely to have inflated these figures. However, the intuition stands, showing that complete validation of inbound measurements for load invariant compliance without optimization rapidly becomes intractable.

Using the period partitioning algorithm, the maximum running time of the admission control checks for a new event being scheduled scales not by the length of time all the repeating instances take times the number of scheduled measurements that overlap in time, but by the number of partitions it traverses, the length of their repeating segments (their hyperperiods), and the number of measurement events in each hyperperiod.

5 Admitting and Scheduling Measurement Requests

As previously discussed, our current prototype only considers inbound and outbound bandwidth constraints. Subject to these bandwidth constraints and the no preemption assumption, our goal is to *maximize the number of satisfied measurement requests, without violating load bounds (i.e., while preserving the load invariant)*. We employ two mechanisms to achieve this: (i) conflating "similar" measurement requests, and (ii) estimating measurement tool bandwidth requirements and comparing their sum at any given time against the resource bounds.[4] We discuss these two techniques in this section.

5.1 Redundant Requests

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 for times T_0, T_0+T_p, T_0+2T_p, ..., may leverage one or more measurement events for the same property, that are already scheduled at times $T_0 \pm pm$, $T_0+T_p \pm pm$, $T_0+2T_p \pm pm$, ... We adjust the definition of r from Section 3.1 to include this parameter, making it (P, T_0, T_p, T_c, pm). 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.

[4] Observe that had we known all the requests *a priori*, had perfect estimates of their bandwidth requirements, and had preprocessed them to eliminate redundant requests, then maximizing the number of admitted requests within load bounds at any given time becomes an instance of the NP-complete Knapsack problem.

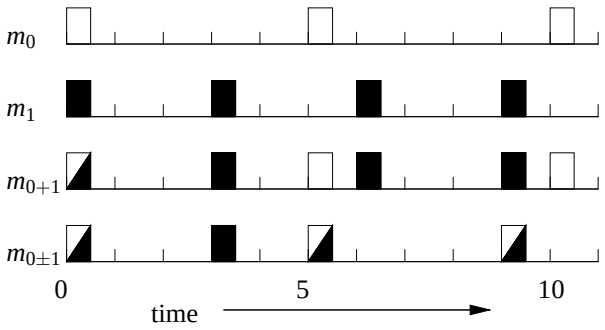

Figure 3: Two measurement schedules, m_0 and m_1, with their combined schedule (labeled m_{0+1}), and the combined schedule with $pm = T_p/2$, labeled $m_{0\pm1}$. The hyperperiod is 15 in this example.

Note that if pm exceeds $T_p/2$, in the absence of other restrictions, one measurement probe can be used to satisfy two requested probe times. We restrict pm to $T_p/2$ for this reason. Fig. 3 depicts two requests for the same property to the same end points, as well as their combined schedule and a collapsed schedule which serves both of these requests with one set of measurements, using $pm = T_p/2$.

The measurement conflation operation is performed by analyzing the partitioned periods created by the algorithm in Section 4. Only the hyperperiod of a partition is examined to find all scheduled events within that partition which can satisfy a new request, and, likewise, the probes in a new request which are unsatisfied within that partition.

Finally, we can utilize secondary properties estimated by a tool (given in Section 3.3), when scheduling a tool that measures a requested quantity as a primary property, is infeasible.

5.2 Admission Control Tests

We investigate two approaches for our admission control test to ensure that the load invariants are preserved. The first approach we devise 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. Therefore, if the specified interval is 1 second, the result of this computation is simply the maximum number of bits transferred in any one second interval. 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, the 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 inbound and outbound traffic of the two end points are used to characterize the tool. Similar to the previous scheme, traffic bounds are expressed as a pair of values, and any measurement which causes the sum of all scheduled measurements at a given time to exceed this value, is rejected.

A possible variation on this second approach estimates the average bandwidth utilization separately for each "phase" of execution of a measurement tool. This applies to measurement tools that have distinct phases with wildly different resource requirements in each phase, such as Pathrate, and not to uniform tools like `ping`. Fig. 4 depicts the resource requirement estimation for the Pathrate tool using this method (measured on an Emulab node). There are four phases and hence four average bandwidth estimates for Pathrate execution: one for the initial spike, one of the ramp which follows, one for the low-bandwidth middle section, and a final average of the high-bandwidth phase at the end. Unfortunately, it is non-trivial to identify the times at which one phase ends and another begins, since these times can be sensitive on the underlying network characteristics.

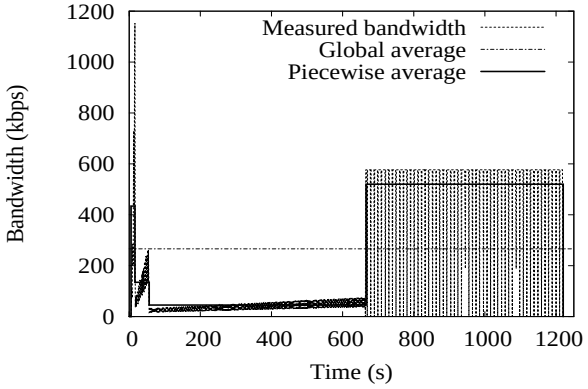

Figure 4: Actual (outbound) bandwidth used by Pathrate in a particular run, along with the single average bandwidth computed over its entire runtime, and the average bandwidth broken into phases.

It is important to note that admission control tests from the connection admission control literature [9, 16, 23] cannot be directly leveraged here due to four primary 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 during different intervals of the measurement tool invocation. 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, whereas the admission control work does not.

6 Prototype and Experimental Evaluation

We have implemented a prototype of the mechanisms discussed in the Sections 4 and 5. In this section, we give the results from a representative set of experiments with our prototype on the Emulab testbed [8]. 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 easily 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.

The experiments in this section are designed to demonstrate four major points. First, we quantify the benefit of applying admission control to measurement traffic under a variety of request workloads. Second, we compare different admission control tests and timescales, with and without methods for eliminating redundant requests. Third, we show that simple estimates of resource requirements of tools such as those described in Section 5 adequately capture the behavior of network measurement tools and allow us to well-utilize our resource budget. Finally, we demonstrate that our admission control tests are sufficiently robust to reasonable variations in network behavior, such as latency and the presence of cross-traffic.

6.1 Experimental Scenario

Fig. 5 gives the topology we use in our experiments. 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 host in each cluster is the "center" of these values. For convenience, we will 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.

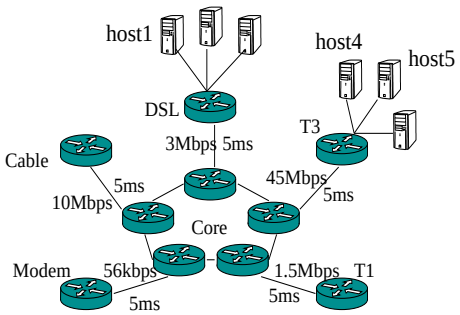

Figure 5: Experimental topology on Emulab. The links in the core network are unmodified 100 Mbps Ethernet links.

Unless otherwise specified, we configure the hosts to use either peak or average bandwidth for admission, and 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 parameters T_0, T_p, and T_c are also randomly selected, such that each request starts and ends within a 45-minute window. Except where otherwise stated, we set *pm* to zero, so that we can easily isolate the impact of each mechanism and interpret the results. Further, we do not utilize secondary properties measured by a tool in these experiments (for the same reason). Each host sets its inbound and outbound bandwidth upper-bounds to 20% of the bandwidth of its access link over the entire experimental duration. Unless otherwise stated, we use 1 second for the duration of the time interval for the peak and average admission control methods.

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 1). 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.

In several of the following scenarios, 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 potentially lead to oversubscription of the modem link bandwidth.

6.2 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 used here is created by taking 10, 25, or 50 random measurement requests, and scheduling them across the DSL, modem, cable, and T3 measurement hosts. Each experiment runs for approximately 45 minutes, or 2700 seconds. We measure the traffic created by measurement tools under this scenario, and compute violations of the specified bounds versus measurements admitted.

Table 2, 3, and 4 give the results of these experiments for 10, 25, and 50 measurement requests, respectively. 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 (rounded up to the nearest integer). The "req'd" (requested) and "failed" columns represent the absolute number of measurement requests involving a given host, and the number of those which were unsatisfied, respectively. The V_{in} and V_{out} columns represent the number of seconds (rounded up) when the inbound or outbound load bounds, respectively, were violated. Within each table, the same set of random workloads was repeated for each admission control method (no admission control, average bandwidth, and peak bandwidth usage).

Note that the average bandwidth usage and number of load bound violations increases steadily with the workload for the scenarios using no admission control. With the average and peak admission control methods, however, the average bandwidth usage peaks very close to the 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 resource budget, for admitting additional measurement requests, especially as the request workload increases.

Table 2: Statistics from an average of ten 10-measurement workloads.

Host	req'd	failed	bw_{in}	bw_{out}	V_{in}	V_{out}
No Admission Control						
T3	5.3	0	158	109	0	0
DSL	4	0	52	123	45	228
Modem	4.9	0	10	13	311	704
Cable	5.8	0	102	67	0	0
Average Admission Control						
T3	5.3	0.3	84	112	0	0
DSL	4	0.7	54	52	1	10
Modem	4.9	2	2	3	48	64
Cable	5.8	1	92	62	0	0
Peak Admission Control						
T3	5.3	0.9	72	85	0	0
DSL	4	1.3	9	38	0	0
Modem	4.9	2.5	2	2	35	39
Cable	5.8	1.3	94	51	0	0

6.3 Redundant Requests and *pm*

Table 5 illustrates the potential savings of measurement tool invocations when $pm > 0$. Each row of the table represents an average of four workloads of a given number of measurement requests from section 6.2. The number of measurement tool invocations in the measurement requests is shown, along with the number of actual invocations which were required to fulfill the requests, for both the case of $pm = 0$ (i.e., exact match required to conflate) and $pm = T_p/2$. It is important to note that many of the additional measurements which are admitted in the scenarios with a larger number of measurement requests are inexpensive, afforded by the larger number of potential requests available to the system. Therefore, the gains in the "Saved" columns represent both additional scheduling freedom afforded by conflation of expensive measurements, as well as potentially large numbers of inexpensive measurements admitted at little or no cost into that additional room. In summary, we find that we can effectively admit additional measurements when there is scheduling flexibility.

6.4 Estimation Accuracy for a Typical Workload

The goal of the next series of experiments is to investigate the accuracy of using peak, average, and piecewise average bandwidth as discussed in Section 5 in more depth, using a typical request workload.

Table 5: Number of requested tool invocations versus actual tool invocations with $pm = 0$ and $pm = T_p/2$, with the percentage of requested invocations saved.

	$pm = 0$			$pm = T_p/2$		
Workload	**Req'd Invocations**	**Invoked**	**Saved**	**Req'd Invocations**	**Invoked**	**Saved**
10	1760	1714	2%	1760	1619	8%
25	3130	3014	3%	3131	2389	23%
50	5307	4842	8%	5660	3431	39%
100	8411	6686	20%	10668	4793	55%

Table 3: Statistics from an average of ten 25-measurement workloads.

Host	req'd	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

Table 4: Statistics from an average of ten 50-measurement workloads.

Host	req'd	failed	bw_{in}	bw_{out}	V_{in}	V_{out}
No Admission Control						
T3	25	0	435	414	0	0
DSL	24.3	0	448	351	1104	713
Modem	25.6	0	26	27	1504	2022
Cable	25.1	0	330	366	26	6
Average Admission Control						
T3	24.8	9.7	250	315	0	0
DSL	23.4	9.8	144	165	190	251
Modem	26.4	17.2	6	7	535	553
Cable	25.4	7.5	334	226	15	1
Peak Admission Control						
T3	25	11.8	186	154	0	0
DSL	24.3	11.5	56	67	0	0
Modem	25.6	17.9	5	5	345	298
Cable	25.1	10	161	179	0	0

To formulate a typical measurement request workload, we create three "roles" for the end points, requiring them to probe various network characteristics. The hosts are divided among those interested in participating in an end-system multicast video stream, online gaming, and an end-to-end network capacity probe (possibly a system administrator). Within each group, pairwise probes are initiated (i.e., n^2 mesh). Direction-sensitive measurements are requested in both directions over each path, while `ping`, our only direction-insensitive measurement, is reduced by the infrastructure to one probe per end-to-end path. The first group of hosts, end-system multicast, requires latency and available bandwidth probes, provided by `ping` and pathChirp, respectively. Latency probes are run every 10 seconds, and available bandwidth every 15 minutes. These hosts are located on the T3, T1, and cable networks. The second group, online gamers, are interested in end-to-end loss and latency. To satisfy these requests, latency probes are scheduled via `ping` every 10 seconds, and loss probes using tulip every 5 minutes. These probes are invoked between measurement hosts on the modem and T1 networks. Network capacity is measured using Pathrate, which is invoked once between the T3 and DSL networks. All probes are scheduled from the measurement hosts starting at time 0, and, as before, complete all repetitions within 45 minutes.

Fig. 6 depicts the measured outbound traffic on the measurement host behind the T3 line, named "host5" in Fig. 5. The plot also shows our estimates of the traffic on this node using the overall average bandwidth scheme. For the peak scheme, we find that the estimated bandwidth usage is much higher than the actual bandwidth seen. This trend is visible in the other experiments that we have conducted. 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 upon 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.

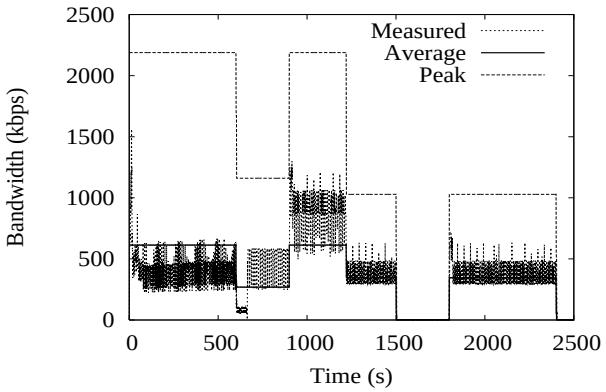

Figure 6: Outbound traffic seen emanating from host5, as well as the estimates using the average and peak bandwidth schemes.

In Fig. 7, we replace the Pathrate estimated bandwidth requirements from Fig. 6 with the requirements using the method that divides traffic into phases and estimates the average for each phase separately (piecewise averages). Clearly, the estimates are much more accurate in this case, demonstrating that using separate estimates for individual phases can have a significant impact for measurement tools which exhibit such wildly differing periods of behavior during a single invocation.

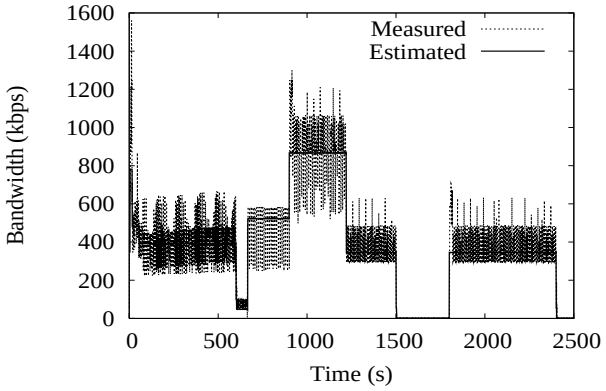

Figure 7: Outbound traffic from host5, with the estimate for Pathrate traffic using an estimator which provides different average bandwidths for each of the "phases" of execution of Pathrate.

6.5 Single-Tool Estimation Accuracy

We ran a series of experimental scenarios which performed random workloads as in the previous section, but where all measurements were restricted to a single property, and thus a single tool. By restricting the schedule in this manner, we minimize intra-tool interactions and can eliminate inconsistencies caused by tools which exhibit undesirable or difficult to quantify behaviors (*e.g.*, the various "phases" of Pathrate). The results of these scenarios for the tulip and pathchirp tools in 25-measurement workloads are shown in Table 6 and Table 7, respectively.

Table 6: Statistics from an average of ten 25-measurement tulip workloads.

Host	req'd	failed	bw_{in}	bw_{out}	V_{in}	V_{out}
No Admission Control						
T3	11.3	0	14	12	0	0
DSL	8.9	0	9	10	0	0
Modem	10.9	0	12	12	1230	1239
Cable	10.2	0	11	12	0	0
Average Admission Control						
T3	11.3	1.6	12	10	0	0
DSL	8.9	1.5	8	8	0	0
Modem	10.9	5.6	7	7	561	609
Cable	10.2	1.2	10	11	0	0
Peak Admission Control						
T3	11.3	2.9	10	9	0	0
DSL	8.9	2.7	7	6	0	0
Modem	10.9	10.9	0	0	0	0
Cable	10.2	3.2	8	9	0	0

Once again, we see that the average bandwidth estimate admits some measurements where the peak bandwidth estimate does not, and consequently generates a larger number of violations. However, in several scenarios (particularly the pathchirp scenarios), it is able to admit more measurement requests than the peak bandwidth estimate without causing any violations at all. Note also that, in some instances, the conservatism of the peak bandwidth estimate mechanism precludes it from admitting any measurements at all on some hosts.

6.6 The Impact of Changing Network Conditions

Our next series of experiments aims to validate that measurement admission control can be performed even under some degree of changing network conditions or underlying characteristics. We will first examine measurement tool behavioral changes in the presence of cross traffic, and then the impact of differing end-to-end delays.

Cross Traffic. In this set of experiments, 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

Table 7: Statistics from an average of ten 25-measurement pathchirp workloads.

Host	req'd	failed	bw_{in}	bw_{out}	V_{in}	V_{out}
No Admission Control						
T3	11.3	0	259	328	0	0
DSL	11.2	0	297	248	409	305
Modem	8.9	0	47	14	336	556
Cable	10	0	273	199	0	0
Average Admission Control						
T3	11.3	4.3	195	257	0	0
DSL	11.2	7.2	156	132	72	35
Modem	8.9	8.9	0	0	0	0
Cable	10	4	201	147	0	0
Peak Admission Control						
T3	11.3	7.4	158	206	0	0
DSL	11.2	9.1	95	105	0	0
Modem	8.9	8.9	0	0	0	0
Cable	10	6.3	157	120	0	0

Table 8: List of transfers used to create cross-traffic for the evaluation of measurement tool characterization in the presence of competing flows.

Traffic type	**Client**	**Server**
Bulk transfer	host4	host7
Bulk transfer	host13	host1
Streaming audio	host13	host4
Streaming audio	host1	host4
Streaming audio	host3	host4
Streaming audio	host13	host1
Web	host10	host1
Web	host6	host9
Web	host6	host3
Web	host1	host6
Web	host15	host6
Web	host7	host13
Web	host9	host3

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. Table 8 summarizes the traffic mix used in these experiments. In the table, "server" and "client" are used in the traditional sense for streaming audio and web traffic, and to indicate the TCP sender and receiver, respectively, for ttcp bulk transfers.

Fig. 8 repeats the same experiment with results depicted in Fig. 6, but with the difference that we inject cross traffic as given in Table 8. We use the average method for the estimate shown in the plot. It can be seen that there is little difference between the measured traffic in Fig. 8 and 6. The largest discrepancy between the measured and estimated values in Fig. 8 is during the final burst period of Pathrate: the burst period is delayed from about time 650 seconds to around 800-900 seconds, and is shorter in overall duration. This demonstrates that admission control methods are still applicable even when the cross traffic conditions when invoking the measurement tools are different from the conditions used for deriving their bandwidth requirements. Overall, we find that the admission control methods are quite robust to changes in cross traffic.

End-to-end Delay. In order to quantify the effects of end-to-end latency on admission control tests, we conducted a series of experiments with 100 randomly-selected requests, using the average estimator. We varied the delay on each of the "arms" of the star topology (the link between the central ring of routers 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 any correlations between these expanding delays and systematic changes in load bound violations, but found none. Again, "violations" means the number of seconds during the scenario where the measured rate exceeds the allowable rate. The reason for violations is that the measurement tool resource requirements were characterized in a different environment than that in which the measurement tool is now being invoked.

Fig. 9 depicts the violations at each delay value for the modem and DSL hosts. As discussed in Section 6.2, the other hosts exhibit very few violations. These findings show that, while the violations differ from scenario to scenario due to differences in the measurement schedule, there is no strong trend related to network delay.

6.7 Estimation Timescales

In this section, we evaluate the impact of the timescale over which measurement tools are characterized and admission control 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 9 shows the percentage of measurements which were satisfied for the various timescales. We utilize a workload of 50 randomly generated requests (the same requests as in Section 6.2), with the results being the average of ten such workloads.

This table illustrates that the average admission control method, as expected, is quite robust to the timescale over which estimations are made. Load bound violations are also consistent across these runs, with varia-

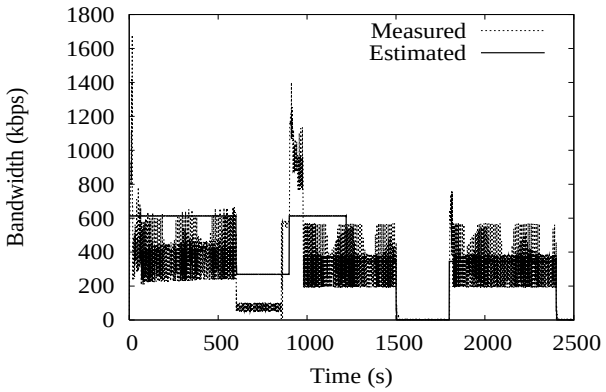

Figure 8: Outbound traffic seen on host5 for the same measurement set as Fig. 6, but with cross traffic in the network. The figure shows that average traffic tool estimates taken from a network with no cross traffic still apply.

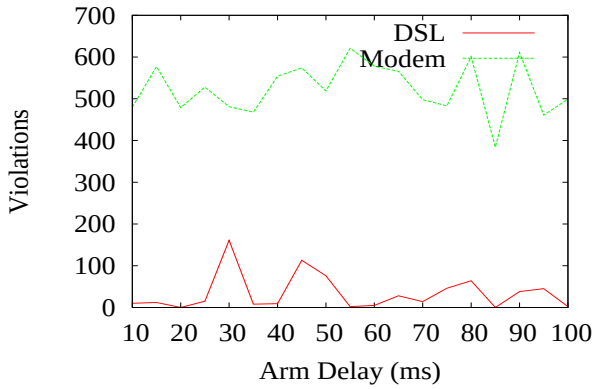

Figure 9: Star "arm" delay versus observed violations at various measurement hosts for increasing one-way delays.

tions between estimation timescales being smaller on average than variations between experiment runs. The peak method varies in its acceptance rate, with the shorter intervals admitting fewer measurements. Violations under the peak method vary correspondingly, with longer intervals displaying more violations than short intervals. 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 more expensive measurements (e.g., using Pathrate) early on, rejecting later measurements of perhaps lower bandwidth requirements. In contrast, the peak bandwidth admission method "fills in" gaps in its schedule with inexpensive and non-conflicting requests. For example, in one particular measurement schedule, the average bandwidth admission controller accepts 8 Pathrate invocations, while the peak bandwidth admission controller admits only 1. This leaves room in the peak bandwidth admission controller schedule to admit 8 more of the less expensive Tulip requests than the average admission controller, which had filled its schedule with Pathrate requests.

Table 9: Measurement request success rates for varying estimation timescales for the average and peak bandwidth admission methods. All figures are averaged over 10 runs of random workloads.

Timescale	Average	Peak
100 ms	59.6%	38.8%
1 s	60.2%	47.2%
2 s	60.2%	46.6%
4 s	60%	51%

7 Related Work

An early measurement infrastructure is the National Internet Measurement Infrastructure (NIMI) mesh [22]. Many of the goals of the NIMI project are similar to this work; however, we focus on a more open architecture, with little or no need for central authentication of individual users, and we endeavor to provide scheduling guarantees which NIMI does not address. Furthermore, NIMI nodes are intended to take only one measurement at a time.

Another early work from Kalidindi *et al.*, Surveyor [13], uses dedicated infrastructure machines to take continuous unidirectional measurements of a few key network properties. Surveyor differs significantly from this work in that it does not provide a broad set of network measurements, and does not allow for them to be scheduled on demand.

Nakao *et al.* propose a "Routing Underlay" architecture [21] which provides typical overlay link performance metrics to applications on an overlay network. In contrast to this routing underlay, our work is intended to expose a wider variety of network measurement information to hosts both within and without the actual measurement mesh, as well as allow requests for specific measurements at stated intervals.

Spring *et al.* propose a public measurement facility, *Scriptroute* [29], which allows individual users, network managers, or researchers to conduct network measurements on a shared measurement mesh, controlled by security and rate-limiting filters. Scriptroute accepts arbitrary measurements at arbitrary times and blindly enforces administrative bandwidth and security filters, po-

tentially invalidating measurements, before the probes even leave the measurement host. Scriptroute lets individual users select, configure, invoke, and use the measurement tools. Our framework eases some of this burden on users, and carefully schedules its measurements such that every accepted measurement is guaranteed to be allowed by the bandwidth and security filters in place on the measurement infrastructure. In addition, our service can better utilize its resource budget since it parses and maintains all requests and exploits any similarities among them.

The Scalable Sensing Service (S^3) [31] is a sensing infrastructure for large-scale distributed and federated systems. It provides data from a number of measurement tools which are run continuously on the PlanetLab infrastructure. Unlike our framework, measurement requests cannot be scheduled by arbitrary users on-demand in S^3.

iPlane [19] also performs scalable continuous measurements over a set of Internet paths. As with S^3, iPlane performs no measurements on-demand. Hence, it is unsuitable for troubleshooting transient faults in progress or other tasks which require real-time results. Additionally, we provide more fine-grained administrative control over the load placed on individual measurement hosts and networks.

Calyam *et al.* present a scheduling algorithm substantially similar to that in Section 4, for many of the same reasons [1]. The behavior of their algorithm is different from ours in the event that a measurement cannot be scheduled as requested, as they treat the schedule as a flexible deadline-driven real-time schedule, rather than our more restrictive notion of *pm*. The authors also present a measurement scheduling service which considers administratively defined load bounds in [2]. Their work concentrates on the effects of the scheduler on request fulfillment and delay, whereas we focus on the issues surrounding the administrative bound and its violations and maintenance.

A number of recent studies have considered specific subproblems that arise in network measurement. The redundant measurement problem has been considered in the context of traceroute in [6]. Their work exploits the tree-like structure of routes, and starts probing a path near its midpoint. ProgME [33] defines "flowsets" as arbitrary sets of flows that can be specified for monitoring and queries, such as heavy-hitter identification. Sommers *et al.* [26] design a unified measurement tool, SLAM, that measures delay distribution, jitter, and loss of a path. These studies are complementary to our work and can be integrated within our framework.

8 Discussion and Future Work

In this paper, we have defined a novel framework for admitting and scheduling recurring active measurement requests. Our framework can enable shared measurement infrastructures to schedule requests *safely* and *on-demand*. The framework exploits the tradeoffs along the measurement time flexibility, required accuracy, and measurement tool resource requirements dimensions, in order to maximize satisfied measurement requests without violating a load invariant. We also presented an algorithm for determining the periods of time (hyperperiods) for which it is sufficient to perform the admission control tests. We find that our techniques are effective and robust through experiments on Emulab.

We find that the characterization of measurement tool resource requirements is important to this work. Due to the extremely bursty nature of measurement traffic, peak bandwidth utilization is an over-conservative metric for characterization. We have demonstrated that average bandwidth utilization is a more accurate metric, but that even this may require refinement in some circumstances, to account for greatly varying utilization during different phases of tool execution. A characterization scheme which captures the essence of this burstiness in a more explicit manner may allow better admission decisions to be made.

Our future work directions include both short-term and long-term goals. In the short-term, we plan to explore methods of aggregating traffic envelopes in a similar manner to [23] to accurately predict future measurement loads. We will evaluate these envelope methods for accuracy on-the-wire with real measurement loads. We will also expand our current experiments to include RocketFuel topologies [28] for a more diverse network environment. Finally, we plan to integrate our system with the S^3 service [31] on the PlanetLab platform.

In the long term, we plan to (i) harness the power of inference mechanisms for estimating quantities when directly measuring them would introduce excessive load, and (ii) investigate how to exploit scheduling flexibility to mitigate interference among measurements. We describe this second future work direction in some detail below.

Active network measurements, by the very traffic they use to perform their duties, perturb the network which they are measuring. Therefore, the presence of one measurement tool can influence the result of another. In order to quantify interference among measurements, we must first define what it means for two measurements to interfere. As a conservative definition, we say that a measurements M_1 *interferes* with a measurement M_2 if at least one packet from measurement M_1 and at least one packet from measurement M_2 occupy the same queue

at the same time, with a packet from measurement M_1 preceding a packet from measurement M_2 in the queue. For the purpose of this definition, we consider network links to be the final entry in a queue. While we believe that this definition captures the essence of interference, it does have limitations which another model might address. For example, it cannot represent the indirect interference caused by two measurements simultaneously traversing the same, overloaded physical router through completely independent links and queues.

This model provides us with a working definition of interference, but is impractical. In the absence of a "network oracle" which can provide the precise disposition of arbitrary queues along a network path at arbitrary points in time, an approximation of this metric is required. For example, we can prove that two measurements are interference-free if we know the maximum amount of time that an active measurement packet can "survive" in the network (call this t_{max}), and the measurements are scheduled such that the last packet transmitted by M_1 is transmitted followed by a period of t_{max} in duration before M_2 begins. Rather than determining precisely where and how measurements are going to interfere in the network, measurements could also be scheduled (when allowed by the *pm* parameter) to minimize the *expected* overall packet rate of the completed measurement event schedule at any given time. We plan to investigate this approach in greater depth.

References

[1] CALYAM, P., LEE, C., ARAVA, P. K., AND KRYMSKIY, D. Enhanced EDF scheduling algorithms for orchestrating network-wide active measurements. In *Proceedings of the 2005 IEEE Real-Time Systems Symposium (RTSS)* (2005).

[2] CALYAM, P., LEE, C., EKICI, E., HAFFNER, M., AND HOWES, N. Orchestration of network-wide active measurements for supporting distributed computing applications. *IEEE Transactions on Computers (TOC) 56*, 12 (Dec 2007).

[3] CHU, Y., RAO, S. G., AND ZHANG, H. A case for end system multicast. In *Proceedings of ACM SIGMETRICS* (June 2000).

[4] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Challenges in employing low-latency anonymity. Public draft at http://www.torproject.org/.

[5] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: the second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium* (2004).

[6] DONNET, B., RAOULT, P., FRIEDMAN, T., AND CROVELLA, M. Deployment of an algorithm for large-scale topology discovery. *IEEE journal on selected areas in communications 24*, 12 (December 2006), 2210–2220.

[7] DOVROLIS, C., AND RAMANATHAN, P. Packet dispersion techniques and capacity estimation. *IEEE/ACM Transactions on Networking* (December 2004).

[8] EMULAB/NETBED. http://www.emulab.net.

[9] FERRARI, D., AND VERMA, D. C. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications 8* (Apr. 1990).

[10] FRALEIGH, C., MOON, S., LYLES, B., COTTON, C., KHAN, M., MOLL, D., ROCKELL, R., SEELY, T., AND DIOT, C. Packet-level traffic measurements from the Sprint IP backbone. *IEEE/ACM Transactions on Networking 17*, 6 (December 2003), 6–16.

[11] HOHN, N., VEITCH, D., PAPAGIANNAKI, K., AND DIOT, C. Bridging router performance and queueing theory. In *Proc. of SIGMETRICS* (June 2004), pp. 355–366.

[12] HU, N., AND STEENKISTE, P. Exploiting Internet route sharing for large scale available bandwidth estimation. In *Proceedings of ACM IMC* (2005).

[13] KALIDINDI, S., AND ZEKAUSKAS, M. J. Surveyor: An infrastructure for Internet performance measurements. In *Proceedings of ISOC INET* (1999).

[14] KESSLER, G., AND SHEPARD, S. A primer on Internet and TCP/IP tools and utilities. Internet RFC 2151, June 1997. ftp://ftp.rfc-editor.org/in-notes/rfc2151.txt.

[15] KIM, M. S., KIM, T., SHIN, Y. J., LAM, S. S., AND POWERS, E. J. A wavelet-based approach to detect shared congestion. In *Proceedings of ACM SIGCOMM* (2004).

[16] KNIGHTLY, E. W., AND SHROFF, N. B. Admission control for statistical QoS: Theory and practice. *IEEE Network 13* (Mar. 1999).

[17] LEDLIE, J., PIETZUCH, P., MITZENMACHER, M., AND SELTZER, M. Network coordinates in the wild. In *Proceedings of NSDI 2007* (Apr. 2007).

[18] LIU, C., AND LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM) 20* (Jan. 1973), 46–61.

[19] MADHYASTHA, H. V., ISDAL, T., PIATEK, M., DIXON, C., ANDERSON, T., KRISHNAMURTHY, A., AND VENKATARAMANI, A. iPlane: An information plane for distributed services. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation* (Nov. 2006), pp. 367–380.

[20] MAHAJAN, R., SPRING, N., WETHERALL, D., AND ANDERSON, T. User-level Internet path diagnosis. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)* (Oct. 2003).

[21] NAKAO, A., PETERSON, L., AND BAVIER, A. A routing underlay for overlay networks. In *Proceedings of ACM SIGCOMM* (2003), pp. 11–18.

[22] PAXSON, V., MAHDAVI, J., ADAMS, A., AND MATHIS, M. An architecture for large-scale Internet measurement. *IEEE Communications Magazine 36* (Aug. 1998), 48–54.

[23] QIU, J., AND KNIGHTLY, E. W. Measurement-based admission control with aggregate traffic envelopes. *IEEE/ACM Transactions on Networking 9*, 2 (Apr. 2001).

[24] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM* (Aug. 2001).

[25] RIBIERO, V. J., REIDI, R. H., BARANIUK, R. G., NAVRATIL, J., AND COTTRELL, L. pathchirp: Efficient available bandwidth estimation for network paths. In *Proceedings of the Passive and Active Measurement (PAM) Workshop* (2003).

[26] SOMMERS, J., BARFORD, P., DUFFIELD, N., AND RON, A. Efficient network-wide SLA compliance monitoring. In *Proceedings of SIGCOMM* (2007).

[27] SONG, H., AND YALAGANDULA, P. Real-time end-to-end network monitoring in large distributed systems. In *Proceedings of IEEE COMSWARE* (2007).

[28] SPRING, N., MAHAJAN, R., AND WETHERALL, D. Measuring ISP topologies with Rocketfuel. In *Proceedings of ACM SIGCOMM* (Aug. 2002).

[29] SPRING, N., WETHERALL, D., AND ANDERSON, T. Scriptroute: A public internet measurement facility. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)* (2002).

[30] SPRUNT, B. *Aperiodic Task Scheduling for Real-Time Systems*. PhD thesis, Carnegie Mellon University, Aug. 1990.

[31] YALAGANDULA, P., SHARMA, P., BANERJEE, S., BASU, S., AND LEE, S.-J. s^3: A scalable sensing service for monitoring large networked systems. In *Proceedings of the ACM SIGCOMM Workshop on Internet Network Management (INM)* (Sept. 2006).

[32] YOUNIS, O., AND FAHMY, S. Flowmate: Scalable on-line fbw clustering. *IEEE/ACM Transactions on Networking 13*, 2 (April 2005).

[33] YUAN, L., CHUAH, C.-N., AND MOHAPATRA, P. ProgME: Towards programmable network measurement. In *Proceedings of SIGCOMM* (2007).