Mitigating Interference in a Network Measurement Service

Sriharsha Gangam
Purdue University
Email: sgangam@cs.purdue.edu

Sonia Fahmy
Purdue University
Email: fahmy@cs.purdue.edu

***Abstract*—Shared measurement services offer key advantages over conventional ad-hoc techniques for network monitoring. A measurement service may receive *measurement requests* concurrently from different applications and network administrators. These measurement requests are often served by injecting active network measurement traffic between two hosts. Two active measurements are said to *interfere* when the probe packets of one measurement tool are viewed as network traffic by the other. This may lead to faulty measurement readings.**

In this paper, we model the measurement interference problem, and show how to schedule measurement tasks to reduce interference and hence increase measurement accuracy. We propose twelve computationally tractable algorithms that decrease the total completion time (makespan) of measurement tasks, while avoiding interference. Our evaluation shows that the algorithm we refer to as *Largest Area First, Busiest Node First - Earliest Interval Schedule* (LAFBNF-EIS) has a mean makespan of about 5% more than the theoretical lower bound over our set of measurement workloads. [1]

I. Introduction

Active measurements are frequently triggered by network administrators and applications to monitor characteristics like delay, loss, and available bandwidth between two nodes in a network. The measurements can then be used to diagnose failures or anomalies, select servers, adapt traffic rates, or select overlay routes, e.g., in a cloud infrastructure or a P2P application. Several network measurement services [14], [22] are used by administrators and applications. A *measurement request* is issued by the administrator or application to the measurement service when it needs to take a measurement (e.g., latency). The measurement service processes measurement requests by triggering tools to measure network properties between *measurement nodes*. The measurement nodes are the endpoints of a measurement.

Active measurement tools inject probe packets into the network. For example, tools like Pathchirp [15] and Pathrate [7] typically send a significant number of probe packets to estimate available bandwidth and capacity, respectively. If the measurement nodes are simultaneously taking multiple measurements, the measurement readings can be faulty. This is because probes of one tool can cause delay or loss of probes belonging to another tool. In other words, the probe traffic of one measurement tool is misinterpreted as network data by the other tools. For example, Pathchirp [15] measures the available bandwidth by increasing the rate of probe packets until congestion is detected. When two instances of Pathchirp probes interfere, congestion is reached much earlier, causing large errors in the measurement. Experiments in [4] suggest that interference among bandwidth measurements is significant. Croce et al. [6] give evidence of strong interference among available bandwidth tools.

Measurement services must schedule active measurement tasks to reduce interference. For periodic network monitoring that is pair-wise across a large number of nodes, the demand on network resources can be significant. Consider the Pathrate tool [7], which includes multiple phases, taking up to 30 minutes for completion. Assume a Pathrate measurement is required every 4 hours between every pair of nodes, i.e., $\frac{n(n-1)}{2}$ measurements are needed, where n is the number of nodes. To avoid interference at the nodes and their edge links (ignoring potential bottlenecks within the network), suppose that each node decides to take a single measurement at a time. Every 30 minutes (duration of Pathrate), at most $\frac{n}{2}$ measurements can be taken because each node can participate in at most one measurement at a time. This implies that the $\frac{n(n-1)}{2}$ measurements require that $(n-1) \times 0.5$ hour ≤ 4 hours. The maximum number of nodes that can satisfy this scheduling criterion is $n = 9$. For larger n and other measurement tools, resources required by active measurements become a bottleneck. The measurement service must identify measurements which interfere when taken simultaneously, and create an efficient non-interfering measurement task schedule.

The objective of this paper is to tackle the interference problem for measurement services by carefully scheduling measurements. Intuitively, the bandwidth consumed by interfering measurements at the bottleneck determines the extent of measurement interference. Based on this interference model, the problem of reducing the total scheduling time for non-interfering measurement tasks can be transformed into a *two-resource constraint packing* problem. We show that even special cases of this packing problem are NP-hard, and we propose and evaluate twelve heuristics to tackle the problem under different measurement workloads.

The remainder of this paper is organized as follows. Sections II, III and IV give our model, assumptions and problem definition. Section V proposes twelve algorithms to schedule

[1]This work has been sponsored in part by Hewlett-Packard, and GENI project 1723. We would like to thank E. Blanton, S. Banerjee, P. Sharma and P. Yalagandula for several discussions on this work.

978-1-4577-0103-0/11/$26.00

measurement tasks. Section VI compares these algorithms under several measurement workloads. Section VII summarizes related work and Section VIII concludes the paper.

II. Understanding Measurement Interference

The problem of interference has been studied for several decades in the context of wireless networks, e.g., [9]. Researchers have proposed opportunistic scheduling approaches to adapt to channel conditions. Our problem bears some resemblance to the wireless scheduling problem, but with key differences. In our problem, every active measurement tool injects measurement probes into the network. These probes are processed based on their arrival times to derive requested properties. Accurate measurement of arrival times of the probes is crucial. For example, Spruce [20] sends probe pairs to a receiver and uses the increase in the gap between the pair to determine available bandwidth. Measurements are inaccurate if these probes are delayed or dropped due to measurement probes from another measurement instance. Thus, it is important to take active measurements at different times. We say that a measurement probe packet A is interfering with another measurement probe packet B if A is delayed or dropped because of B. Measurement instances are said to interfere if at least one of their probe packets interfere.

One method to avoid interference among measurement tools is to check if the paths taken by measurement probes share any router. In [4], [8], two measurements are considered interfering when they share a common sub-path. On an Internet scale, a common measurement sub-path can be difficult to identify since the network topologies and routes are not always available (some routers do not respond in trace-routes, and alias resolution of interfaces of the same router remains a difficult problem). Though techniques as in [17] can be used to detect shared sub-paths, the overhead of using these tools is high. Additionally, route changes necessitate periodic topology inference.

The impact of interference is likely to be highest at a shared bottleneck link. Today, the access links in edge networks are typically where the bottleneck lies [12]. The degree of interference is thus related to the bandwidth consumed by other measurement tools at the access link. We use this intuition to build an interference model and develop measurement task scheduling algorithms.

III. System Model

Each measurement request R_i (where i is an integer that ranges from 1 to the number of requests) can be expressed as a vector $R_i = (Src_i, Dst_i, C_i, D_i)$. This indicates that a measurement request R_i will use a measurement tool to estimate a property (e.g., packet loss) between source node Src_i and destination node Dst_i. This measurement tool is estimated to take D_i seconds to complete its task, at a cost of C_i, which can represent the bandwidth consumed by the measurement tool.

TABLE I
Estimated costs (in Kbps) and durations (in seconds) of measurement tools based on bandwidth consumption for a threshold $H_n = 1,000$ Kbps.

Tool	Bandwidth	Duration	C_i	D_i
Ping	$< 1\ Kbps$ per probe	1 RTT	1	5
Pathrate	function of end-to-end bandwidth	$\sim$ 20 minutes	1,000	1,200
Pathchirp	$\sim$ 1 Mbps peak	$\sim$ 10 minutes	1,000	600
Tulip	$\sim$ 20 Kbps peak	linear function of delay	20	300

A. Measurement Interference Model

Interference among measurement tools can be mitigated by imposing a limit on the total measurement cost, e.g., total bandwidth consumed by all measurement tools on the access links of measurement nodes. This limit can be set by a network administrator to bound the resources used by measurement tools, as in our prior work [2]. Each measurement node n therefore has a threshold, H_n, which represents the maximum amount of measurement probe traffic tolerated on its access link.

Cost assignment for different measurement instances of a measurement tool can depend on the degree of interference tolerated. For example, assigning a measurement cost $C_i = H_n$ for all measurement tools ensures that a node does not run more than one measurement tool at a time. A measurement cost of $C_i = 0$ accepts any kind of interference. Each instance of a measurement tool is assumed to incur the same cost for the same time duration at both the source and destination of the measurement (as discussed below).

Our model is general enough to capture requirements for admission control and interference mitigation for all types of measurement tools based on cost (in terms of bandwidth consumption). The model does not make any assumptions on the nature of the measurement tools used, so that it can accommodate future measurement tools.

B. Measurement Costs and Durations

An important component of the interference mitigation problem is the cost and duration estimation for measurement tools. In our previous work [2], we quantified the bandwidth consumption and duration of several popular measurement tools for several network configurations. Table I gives sample assignments of cost (C_i) and duration (D_i) based on our results in [2]. For example, the C_i for Pathrate is H_n (e.g., 1 Mbps) as it can use up the entire bandwidth limit. The cost values of the remaining tools are based on their peak bandwidth consumed. The threshold values, H_n, should be greater than the maximum cost (C_i). The duration of ping of 5 seconds is used to account for any timing inaccuracies due to clock drift and latency during the measurement.

We can also include the effects of computation load, not just bandwidth consumption, at end hosts to determine the cost and duration values for a measurement instance. Song et al. [18] studied the impact of end host load (due to

multiple measurement instances) on the accuracy of results from measurement tools.

When a measurement task is scheduled to start at a time T, the exact times at which the source and destination nodes actually participate will be different. To account for the delay of the path between the measurement nodes, one should use a slightly longer estimate of the measurement duration than the actual value. Suppose that a measurement tool is invoked almost simultaneously at time T_1 between two measurement nodes, Src_i and Dst_i. We assume that the probes sent by Src_i at time T_1 will arrive at Dst_i at a later time $T_1 + \delta$, where δ is mainly determined by the sum of the propagation and queuing delays between the two nodes. A conservative estimate of δ can be added to the duration D_i.

C. Summary

Each measurement node is associated with a bottleneck (typically at the access link) which is the primary location of active measurement interference errors. Each measurement tool instance has a cost C_i and duration D_i. Scheduling a measurement between two nodes Src_i and Dst_i incurs a cost of C_i at both endpoints. Each measurement node has configurable threshold H_n, which represents the maximum total measurement cost acceptable to the endpoint at any time. Measurements are assumed to be largely non-interfering if the total cost of all measurement tools at an endpoint does not exceed its threshold H_n.

IV. Problem Statement

Our aim is to find a feasible schedule that will rapidly complete the required measurements without compromising accuracy. Taking measurements in a serial order can be time-consuming. Song et al. [18] reported that all pair Pathrate [7] measurements between 500 PlanetLab nodes take 31 hours when measurements are serialized. Thus, efficient scheduling methods are required to incorporate parallelism in taking multiple measurements without interference.

Given a set of measurement requests R_i, our objective is to schedule measurements in order to minimize the time to complete all measurements, *makespan* [5], such that the cost incurred by measurements at every node n is within the threshold H_n at all times. We call this the two-resource constraint packing problem. There is no priority among the tasks and the measurements cannot be preempted. Fig. 1 gives an example schedule of measurements satisfying threshold constraints. Note that the cost of a measurement is incurred at both the source Src_i and destination Dst_i measurement nodes. This is indicated by rectangles of the same fill shape. The objective is to find an efficient schedule to complete the measurement tasks.

Consider the special case of the problem when $C_i = H_n = H, \forall i, n$ and duration of all tasks $D_i = T, \forall i$. The problem of minimizing the makespan is equivalent to finding the chromatic index (edge chromatic number) of the *measurement request graph* [3] corresponding to the set of measurement tasks to be scheduled. Each vertex in a measurement request graph corresponds to a measurement node, and a measurement request between the nodes corresponds to an edge [3]. The optimal makespan is equal to the chromatic index times T. Even this special case of the problem is NP-complete [11].

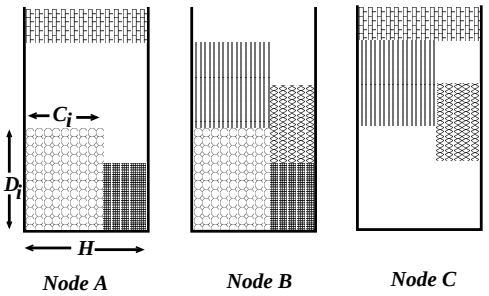

Fig. 1. An example schedule of measurements among three measurement nodes. Rectangles with the same fill shape correspond to the cost of a measurement tool incurred on measurement nodes. The vertical axis represents time and the horizontal axis is the cost of a measurement tool. The threshold H_n represents the maximum measurement cost that can be accommodated at a node.

The two-resource constraint packing problem is a general case of the file transfer scheduling problem in [5], where H_n corresponds to the limit on port numbers, and each measurement task is analogous to a file transfer with all cost values $C_i = 1$. The file transfer scheduling problem was shown to be NP-hard [5].

Our problem is also related to the NP-hard multi-dimensional vector packing problem [13]. The vector packing problem is concerned with minimizing the number of unit sized multi-dimensional bins which can accommodate vectors. Our problem corresponds to the case of $(n + 1)$ dimensions; one dimension for time and a threshold dimension each of the n nodes limited by the corresponding H_n (threshold) values. Each measurement task has $(n + 1)$ dimensions with three non-zero dimensions. One time dimension corresponds to the duration of the task and the other two dimensions indicate the cost at the source and destination nodes. Our problem requires that there is a single bin with infinite length across the time dimension.

V. Approximation Algorithms

We have implemented the optimal algorithm for the two-resource constraint packing problem, generating all possible schedules. The optimal schedule for a measurement request graph of 5 end nodes and 10 measurement requests (i.e., measurement tasks or edges in the graph) takes more than three days without coming to a halt on a 1.86 GHz processor. In contrast, one of the best approximation algorithms we will describe in this section (LAFBNF-EIS) only takes 54 seconds for a measurement request graph with 500 nodes and $124,750$ requests.

We now propose two basic approximation algorithms and extend them to reduce the makespan.

A. Earliest Interval Schedule (EIS)

In the first approximation algorithm, each node maintains a *resource utilization list*. A resource utilization list is a

list of contiguous time intervals with corresponding *available resources* for each interval. The available resources for an interval determine the maximum cost task than can be accommodated in that interval. Time intervals are said to be contiguous when the end time of an interval is the start time of the next interval in the list. The length of a time interval determines the maximum duration of a task that can be scheduled in that interval. The resource utilization list of every node is initialized with a single element, corresponding to the time interval $[0, \infty)$ with an available capacity equal to the threshold (H_n) of the node. When a task is scheduled during a time period at a given node, the available resources for all intervals overlapping with this time period are updated to reflect the resource consumption of this task. This process may involve creation of at most two new intervals in the resource utilization list of the node. Thus, there can be at most $2|V|$ time intervals at any node, for the case when measurement request graphs do not have multiple edges between vertices (e.g., only Pathrate measurements). In the remainder of this paper, we assume no multiple edges exist between the nodes.

To schedule a task between nodes Src_i and Dst_i, the algorithm scans the resource utilization list at Src_i, and outputs a list of the longest non-contiguous compound time intervals that can accommodate the task. A compound time interval is the union of a set of contiguous time intervals with each having an available resource more than the cost of the task. A compound time interval should have an interval length greater than the duration of the task. Similarly, a list of longest non-contiguous compound time intervals is computed for the destination node Dst_i. Compound interval lists from both the nodes are then processed to obtain an earliest common time interval that can accommodate the task. The time complexity of this algorithm is $O(|V||E|)$, where $|E|$ is the number of measurement requests.

B. Progressive Time Schedule (PTS)

In the second approximation algorithm, initially (at $t = 0$) and after every task is scheduled to complete, the list of all tasks is scanned for tasks that can be scheduled immediately. When a feasible task is found, the task is scheduled to start immediately. When a task cannot be scheduled at the current time, the next task from the list is processed. If none of the remaining tasks are feasible, the process is repeated at the earliest time when a task is completed and resources are freed. There is no point in time when resources are available and a task is delayed (i.e., this is a work-conserving algorithm). Algorithm 1 outlines the operations. The time complexity of this algorithm is $O(|E|^2)$. Compared to EIS, the PTS algorithm is more expensive and, in some sense, aggressive, so we use it to compute worst case makespan bounds later in this section.

Algorithm 1 Progressive Time Schedule (PTS)

```
TS ← Task Set ; S ← ∅
time ← 0
while TS ≠ ∅ do
  for each τ ∈ TS do
    if isFeasibleNow(τ) then
      start(τ) ← time
      end(τ) ← start(τ) + Duration(τ)
      TS ← TS − {τ}
      S ← S ∪ {τ}
    end if
  end for
  time ← min_{τ∈S} {end(τ) : end(τ) > time}
end while
```

C. Ordering Heuristics

The order of measurement tasks processed by the EIS and PTS algorithms plays an important role in determining the makespan of the schedule generated. Any given schedule of tasks can be represented by a total order of tasks based on their start times. For a given set of measurement requests, PTS may not give an optimal solution, regardless of the order in which the requests are given. This is due to its work-conserving nature: the optimal schedule (total order) of tasks may be non-work-conserving. An example can be found in [5]. In the case of EIS, an enumeration of input order of tasks exists that will result in an optimal total order of tasks. This motivates us to consider several heuristics for the processing order of tasks.

Theoretical results in [5] indicate that the worst case makespan bound is lower if the requests are processed in increasing order of duration. Similarly, Corollary 2 below suggests that processing measurement requests in decreasing order of cost can reduce the makespan. Based on this intuition, we propose to extend each of basic EIS and PTS algorithms as follows:

1) *"Costliest" Task First-Earliest Interval Schedule (CTF-EIS)*: Process requests in decreasing order of costs.
2) *Longest Task First-Earliest Interval Schedule (LTF-EIS)*: Process requests in decreasing order of duration.
3) *Largest Area First-Earliest Interval Schedule (LAF-EIS)*: Process requests in decreasing order of the *area* of the measurement request. The *area* is defined as the product of cost and duration.
4) *Busiest Node First-Earliest Interval Schedule (BNF-EIS)*: Process requests in decreasing order of a metric, *busy*. The value of *busy* of a node u is given by $(\sum_{\tau_i \in TS(u)} D_i C_i)/H_u$, where $TS(u)$ is the set of all tasks (requests) that have node u as the source or destination. The busy metric associated with a task is the maximum value of the busy metrics of its source and destination end nodes.
5) *Largest Area First, Busiest Node First-Earliest Interval Schedule (LAFBNF-EIS)*: Process requests in decreasing order of area. When the areas of two tasks are equal, the busy metric is used to to assign priority.

Similar to the above heuristics for EIS, we have CTF-PTS, LTF-PTS, LAF-PTS, BNF-PTS, LAFBNF-PTS for the PTS algorithm. We will consider *offline scheduling* (i.e., when the

complete set of requests is available for processing at the time the algorithm is triggered) in the remainder of this paper. The algorithms can be easily adapted for online processing at the cost of reduced performance.

D. Bounds on makespan

We first give a lower bound on the optimal makespan.

Lemma 1. *Let $OPT(G)$ be the optimal makespan for a given measurement request graph $G(V, E)$.*

$$OPT(G) \geq \max_{u \in V} \left\{ \frac{\sum\limits_{\tau_i \in TS(u)} D_i C_i}{H_u} \right\},$$

where $TS(u)$ is the set of all tasks that have node u as the source or destination.

Proof: Let $MS(u)$ denote the minimum finish time of the last task scheduled at vertex u. $OPT(G) \geq \max_{u \in V} \{MS(u)\}$, where

$$MS(u) \geq \frac{\sum\limits_{\tau_i \in TS(u)} D_i C_i}{H_u}.$$

Equality occurs when the vertex u has the bandwidth threshold H_u completely utilized for the entire duration. Thus, we have,

$$OPT(G) \geq \max_{u \in V} \left\{ \frac{\sum\limits_{\tau_i \in TS(u)} D_i C_i}{H_u} \right\}.$$

■

Let $PTS(G, S)$ denote the makespan of a *Progressive Time Schedule* S for the measurement request graph $G(V, E)$. Let D, C, u, v be the duration, cost, source, and destination of the last task τ of schedule S. A node u is said to be busy at a given time when it does not have enough free resources to schedule the last task τ. Since the task τ is scheduled last, at least one of the nodes u and v must be busy until the task τ has been initiated. Let $C_{busy}(u, S)$ denote the minimum occupied resource at node u that does not let the last task have an earlier start time. Note that when the node u is busy, the available resource ($H_u - C_{busy}(u, S)$) must be less than C. Otherwise, the last task would have been scheduled at some earlier time according to the PTS algorithm. We use the above notation to derive the following upper bounds on the makespan for PTS and CTF-PTS.

Theorem 1.

$$PTS(G, S) \leq \left(\frac{H_u}{C_{busy}(u, S)} + \frac{H_v}{C_{busy}(v, S)} \right) OPT(G) + D \left(1 - \frac{C}{C_{busy}(u, S)} - \frac{C}{C_{busy}(v, S)} \right).$$

Proof: From the definition of $C_{busy}(u, S)$, it follows that the time period during which the task τ cannot be scheduled because u is busy is at most

$$\frac{\sum\limits_{\tau_i \in TS(u)} D_i C_i - DC}{C_{busy}(u, S)}.$$

Similarly, the time period for which v is busy to schedule the task τ is at most

$$\frac{\sum\limits_{\tau_i \in TS(v)} D_i C_i - DC}{C_{busy}(v, S)}.$$

Hence, the total time for which at least one of u and v is busy can be at most their sum. Thus,

$$PTS(G, S) \leq \Big(\sum_{\tau_i \in TS(u)} D_i C_i - DC \Big) / C_{busy}(u, S) + \Big(\sum_{\tau_i \in TS(v)} D_i C_i - DC \Big) / C_{busy}(v, S) + D.$$

$$PTS(G, S) \leq H_u / C_{busy}(u, S) \left(\frac{\sum\limits_{\tau_i \in TS(u)} D_i C_i}{H_u} \right) + H_v / C_{busy}(v, S) \left(\frac{\sum\limits_{\tau_i \in TS(v)} D_i C_i}{H_v} \right) + D \left(1 - C / C_{busy}(u, S) - C / C_{busy}(v, S) \right).$$

Using lemma 1, we have

$$PTS(G, S) \leq \left(\frac{H_u}{C_{busy}(u, S)} + \frac{H_v}{C_{busy}(v, S)} \right) OPT(G) + D \left(1 - \frac{C}{C_{busy}(u, S)} - \frac{C}{C_{busy}(v, S)} \right).$$

■

Corollary 1. *Let $PTS_{CC}(G, S)$ denote the makespan of a schedule S computed using the PTS algorithm for a measurement request graph $G(V, E)$ when all the measurement tasks have equal costs.*

$$PTS_{CC}(G, S) \leq 2 \times OPT(G) + D.$$

In other words, the problem is simpler when the measurement costs are equal and a tighter bound on the makespan provided by PTS algorithm exists.

Proof: When all task costs are equal to C , the threshold H_i of each node i can be replaced with $H_i^* = C \times \lfloor H_i / C \rfloor$. This is because the available resource of $H - H_i^*$ at node i cannot be utilized by any task. For a given node i to be busy, we must have $C_{busy}(i, S) = H_i^*$ because the threshold H_i^* is

filled in integral multiples of C. Thus, from Theorem 1, we have,

$$PTS_{CC}(G,S) \leq 2 \times OPT(G) + D.$$

■

This is consistent with the result in [5] where all costs have a value of 1.

Corollary 2. *Let $CTF\text{-}PTS(G,S)$ denote the makespan of a schedule S computed using the CTF-PTS algorithm for a measurement request graph $G(V,E)$.*

$$CTF\text{-}PTS(G,S) \leq 4 \times OPT(G) + D.$$

In other words, for general task configurations with varying cost metrics, processing measurement tasks in the decreasing order of costs can reduce the makespan of the schedule.

Proof: As mentioned earlier, the available resources at u must be less than the cost of the last task C when u is unable to schedule the last task, i.e., $H_u - C_{busy}(u,S) < C$.

Consider the cost of the first task C_f scheduled at u that causes the node to become busy for the last task τ. Since the tasks are scheduled according to the costliest task first, we must have $C_f \geq C$. After task C_f completes, there is a possibility that the last task is scheduled, unless there is another task at the node with cost higher than C. Thus, it can be seen that whenever a node u is busy for the last task, there is a task with cost higher than C scheduled. Hence we have, $C_{busy}(u,S) \geq C$. Thus,

$$C_{busy}(u,S) \geq C > H_u - C_{busy}(u,S),$$

which results in $C_{busy}(u,S) > H_u/2$. Using a similar argument, the result holds for node v as well (i.e., $C_{busy}(v,S) > H_v/2$). Using these values in Theorem 1, we have

$$CTF\text{-}PTS(G,S) \leq 4 \times OPT(G) + D.$$

■

VI. Evaluation

A. Measurement Workloads and Thresholds

The workloads we use in our evaluation can be classified based on (1) the structure of the measurement request graph, (2) the distribution of cost thresholds at the measurement end nodes, and (3) the distribution of costs and durations of measurement task requests.

Measurement request graph:

(1) Complete graph: The set of measurement requests includes all pair-wise measurements. This is a typical scenario for network administrators who wish to evaluate all paths in their network to detect any potential misconfigurations or failures.

(2) UUSee graphs: The measurement request graph is based on peer connections among nodes in the UUSee live streaming video service. The node degree distribution and clustering properties of the UUSee service have been studied in [21]. These node properties were used to construct graphs with $2,500$ vertices and about $53,000$ edges. The clustering coefficient and the average path length in these graphs are close to 0.25 and 2.3 respectively. More information on the UUSee graph generation can be found in [3]. This scenario represents measurements conducted by a typical peer-to-peer application to determine the best peers to download data from.

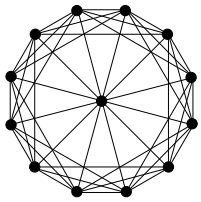

Fig. 2. Extended wheel graph for 13 nodes and heterogeneity 0.25. The degree of the center node is 12 and that of a non-center node is 7.

(3) Extended wheel graph: In this graph, there is an edge between the center vertex and every other vertex (non-center vertex). Additional edges are constructed between non-center edges in the following manner. Each non-center node connects to $\lceil (n-1)h \rceil$ non-center nodes along the circumference of the wheel in an anti-clockwise manner, where h is the heterogeneity of the graph and n is the number of nodes in the graph. The heterogeneity of an extended wheel graph is therefore a measure of uneven degree distribution between the center node and non-center nodes. Fig. 2 is an extended wheel graph for 13 vertices and heterogeneity 0.25. Note that when $h = 0.5$, this becomes a complete graph (possibly with some double edges). For a low value of h, it becomes a star graph or a wheel graph. The extended wheel graph is used to represent measurement request graphs which have a heterogeneous distribution of the number of tasks (load) on the center node. This setup is typical when a network administrator or application is predominantly (but not necessarily exclusively) interested in properties to a particular node.

Node cost threshold:

The values used for the threshold H_n can be divided into the following categories: *(1) Constant:* The threshold values for every node is the same. $H_n = 1,000$ for all nodes. *(2) Random:* H_n of each node is chosen uniformly randomly among multiples of $1,000$ which lie between 999 and $5,001$.

Task cost and duration:

We use the following cost and duration configurations:

(1) Constant Cost-Constant Duration: The cost and duration values for every task take a constant value. $C_i = 1,000$ and $D_i = 1,800$ for all tasks.

(2) Bandwidth-based: The cost and duration values are based on bandwidth usage and duration of measurement tools. Every task is assigned cost and duration values chosen from one of the entries in Table I.

(3) Random Cost-Constant Duration: The duration of all tasks is assigned the same value, while the cost takes random values. $D_i = 1,800$ for all tasks. C_i is chosen uniformly randomly between 10 and $1,000$.

(4) Constant Cost-Random Duration: The cost of every task is assigned the same value while the duration takes random values. D_i is chosen uniformly randomly between 10 and $1,000$. $C_i = 500$ for all tasks.

(5) Random Cost-Random Duration: Both the cost and duration of every task are assigned random values. D_i and C_i are chosen uniformly randomly between 10 and $1,000$.

B. Performance of Scheduling Algorithms

It is important to conduct a fair comparison of the values of makespan among the heuristics across multiple runs and data sets. To achieve this, we normalize the makespan (MS) with the lower bound (LB) for each experiment. Let OPT denote the optimal makespan for a given task workload and node configuration. Since $MS/OPT \leq MS/LB$, if the MS/LB values are close to 1, this is an indication that the approximation algorithms are working well. The lower bound of the makespan is computed using lemma 1. For each type of data set, the experiments are repeated 30 times.

We find that most of our approximation algorithms generate schedules with low makespan. The mean, median, and maximum values of MS/LB for one of the best heuristics (LAFBNF-EIS) are 5%, 2% and 36% more than theoretical lower bound values over all the workloads. In this section, any reference to the ordering heuristics CTF, LTF, LAF, BNF, and LAFBNF applies to both the EIS and PTS-based algorithms, unless otherwise specified.

1) Complete Graph: When the workload is a complete graph, the performance improvement of the ordering heuristics LTF and CTF over the basic EIS and PTS algorithms depends on the type of task workload. In the *Constant Cost-Random Duration* task set, the tasks are distinguishable by their duration times. As expected, LTF performs better in this case, as it gives preference to longer tasks. Similarly, CTF performs better for the *Random Cost-Constant Duration* task set. The LAF and LAFBNF heuristics which give preference to tasks with a larger area (cost duration product) perform on par with the CTF and LTF heuristics for both data sets. Further, when both the cost and duration are random, LAF and LAFBNF perform better than both CTF and LTF.

2) Extended Wheel Graph: Fig. 3 depicts the MS/LB values for an extended wheel graph with 300 nodes and heterogeneity 0.25 with constant node threshold values. The figure illustrates the distribution of MS/LB values of different algorithms using violin plots [10] from the lattice package [16] in R. A violin plot is constructed using a box-plot with an additional kernel density plot on each side of the box plot. The widest part of a violin corresponds to a larger set of points in that region. If the violins are unimodal and symmetric (as in our case), the widest part also corresponds to the median.

When the task configuration is *Constant Cost-Random Duration*, and *Random Cost-Constant Duration*, the LAF and LAFBNF heuristics work well (better than CTF and LTF). This is seen in Fig. 3 where the violins which signify the distributions of MS/LB are closer to 1 compared to other heuristics. However, when the task configuration is *Constant Cost-Constant Duration* (not shown) and *Bandwidth-based*, BNF and LAFBNF show improved performance over other heuristics.

In extended wheel graphs, the center node is overloaded with measurements. If the center node is kept busy (no free available resources), all measurements among non-center nodes typically complete before the center node becomes free. If the center node waits for non-center nodes, the makespan increases. The BNF and LAFBNF heuristics prioritize tasks which have the center node as the measurement source or destination. Thus, they work well with extended wheel graphs. The LAFBNF heuristic, which incorporates the graph structure of measurement requests (the busy metric), as well as the cost and duration of tasks, in computing task priorities exhibits the best performance for all workloads.

We also note that the MS/LB ratios of a given heuristic vary across cost-duration configurations. When the configuration is based on random cost, the ratios are higher, because a larger number of "holes" (unused resource intervals) are expected in the schedule. When the configuration is *Constant Cost-Random Duration*, the MS/LB values are close to 1, indicating that the heuristics yield the best performance in this case. This behavior is consistent with the worst case bounds in corollary 1 being lower than corollary 2 for the general case.

We also compute the average response times of the tasks in a schedule. The response time of a task is defined as the time taken to complete the task from the start of the schedule. Our heuristics aim to reduce the makespan by giving preference to longer and larger tasks, and tasks on busy nodes. As expected, the algorithms that effectively reduce the makespan have longer response times. Fig. 4 illustrates the response times for different heuristics for the extended wheel data set. For a fair comparison of the heuristics across different experiments, we use the ratio of the average response time (ART) and the lower bound (LB) as a comparison metric. Note that the basic BNF heuristics, which do not deal with cost or duration of individual tasks in assigning priority, have low response times. BNF also has lower makespan schedules for the extended wheel graph, making it preferable over LAFBNF in such graphs.

In the case of UUSee graphs, most heuristics perform equally well when the workload is *Bandwidth-based*. For other task configurations, the LAF and LAFBNF heuristics exhibit the best performance. The plots are not shown due to space constraints.

Table II lists the best heuristics for a given task type and measurement request graph. It can be seen that the performance of EIS and PTS-based heuristics is similar for most of the workloads. EIS-based heuristics tend to perform better with the *Random Cost-Random Duration* task configuration. PTS-based heuristics tend to perform better with the *Constant Cost-Random Duration* configuration. The LAFBNF-based heuristics perform consistently well over all graph topologies and task configurations. These results remained consistent in our evaluation for measurement request graphs of other sizes and node threshold configurations.

VII. Related Work

Our problem boils down to a task scheduling problem, where each measurement task consumes equal resources at the

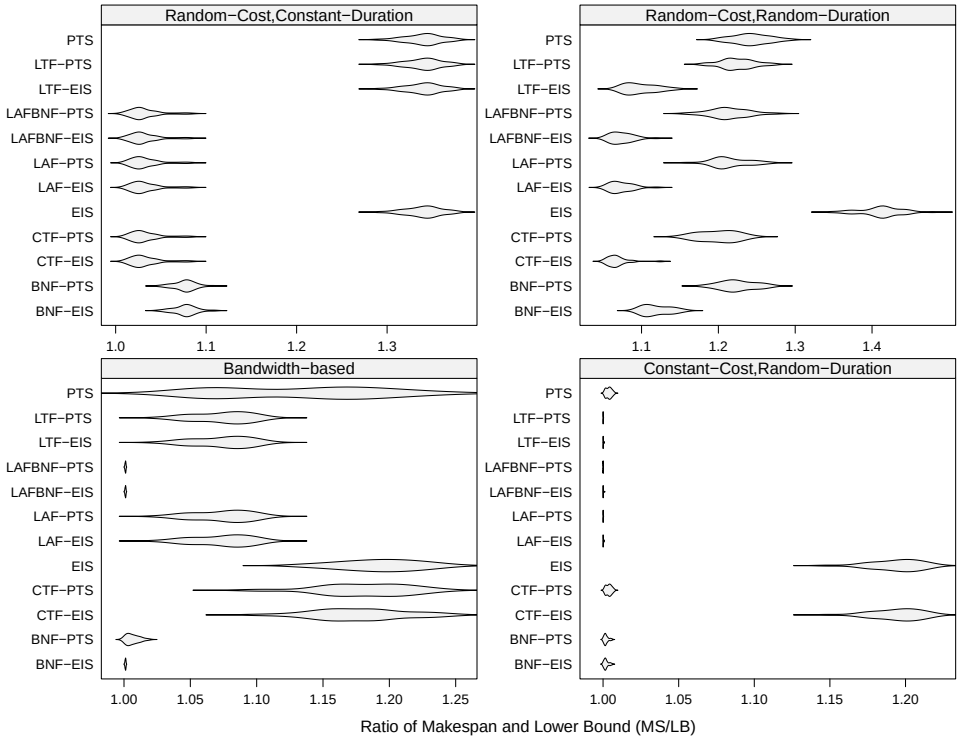

Fig. 3. Makespan comparison of different heuristics for different workloads following an extended wheel graph.

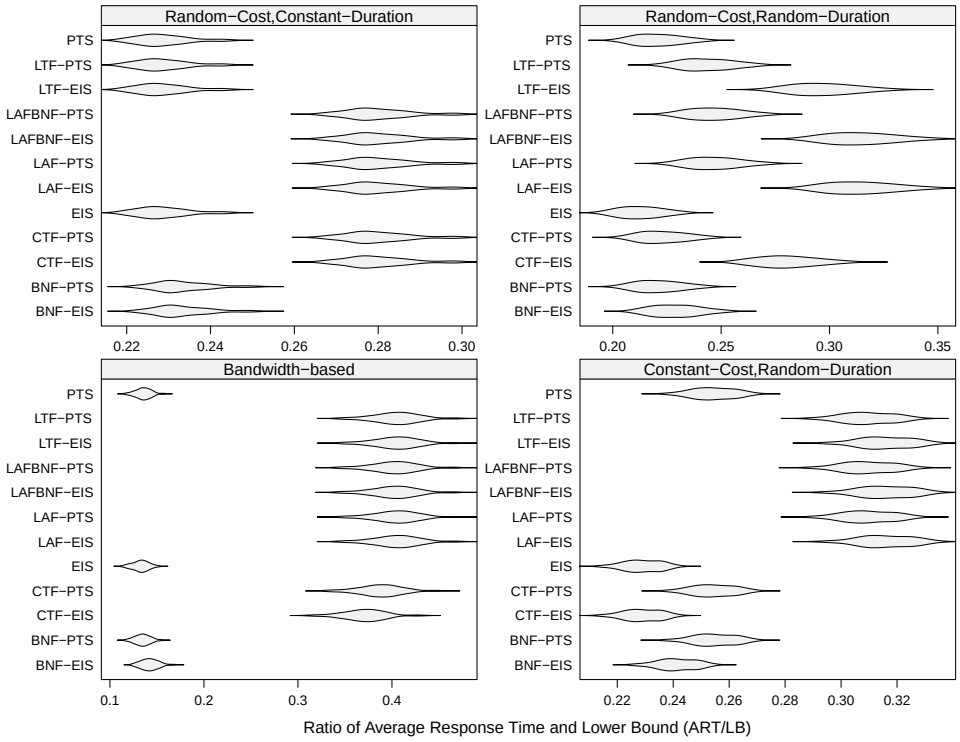

Fig. 4. Average response time comparison of different heuristics for different workloads following an extended wheel graph.

TABLE II
HEURISTICS GIVING SMALLEST MAKESPAN FOR DIFFERENT TASKS AND MEASUREMENT REQUEST GRAPHS. * INDICATES BOTH EIS AND PTS VERSIONS.

Task Type	Complete (300 nodes)	Extended Wheel (300 nodes)	UUSee (2,500 nodes)
Constant Cost-Constant Duration	ALL	LAFBNF*, BNF*	ALL
Bandwidth-based	LAFBNF* , LAF*, LTF*	LAFBNF*, BNF-EIS	ALL except EIS, PTS, BNF-PTS
Random Cost-Constant Duration	LAFBNF*, LAF*, CTF*	LAFBNF*, LAF*, CTF*	LAFBNF*, LAF*, CTF*
Constant Cost-Random Duration	LAFBNF-PTS, LAF-PTS, LTF-PTS	LAFBNF*, LAF*, LTF*	LAFBNF*, LAF*, LTF*
Random Cost-Random Duration	LAFBNF-EIS, LAF-EIS, CTF-EIS	LAFBNF-EIS, LAF-EIS, CTF-EIS	CTF*

two access links of the end points of a measurement during the same time interval. Efficient data transfer among nodes using a shared wireless medium is analogous. Each pair of nodes wishing to communicate can be mapped to a measurement request between a pair of nodes. However, wireless nodes cannot receive messages from more than one node at a time (unless there are multiple antennas), whereas it is possible for a measurement node to take multiple measurements simultaneously.

In [4] and [8], a task conflict graph is constructed to represent interference. A task (a measurement request) is denoted by a vertex and a conflict is represented by an edge between the tasks. Two measurement tasks are said to be conflicting if the measurement paths share a common link or node. In [4], the authors assume that the measurement tasks are associated with deadlines. An earliest deadline first scheduling algorithm (EDF-CE) is then used to schedule measurements. In [8], the tasks are grouped into periods such that all tasks within a period are non-conflicting. The tasks within the same period are executed concurrently. The problem we are solving is more general than the two above-mentioned studies in that we assign different costs to each tool. Additionally, a conflict graph is unsuitable for our case, since tasks can be concurrently scheduled as long as they do not violate resource constraints.

VIII. CONCLUSIONS

We have modeled the interference mitigation problem among active measurements. We address the problem by limiting the total bandwidth consumption of active measurement tools over bottleneck links. We transform this problem of scheduling measurement tasks within bandwidth constraints into a two-resource constraint problem, and show that it is NP-hard. We propose twelve approximation algorithms that aim to reduce the completion time of the set of measurements. We also derive theoretical upper and lower bounds for some of our algorithms.

We conduct a detailed evaluation to compare our algorithms. For most data sets, the time to complete all measurements (makespan) is close to the lower bound. The LAFBNF-EIS heuristic, which considers load on the nodes as well as cost and duration of measurement tasks, consistently produces a low makespan schedule. We also evaluate the response times of measurement requests. We find that heuristics that consider node load, BNF, yield the best results in this case. In our future work, we plan to derive tighter upper bounds for all heuristics. We will also integrate the scheduling mechanisms into the S^3 measurement service [22] deployed on ProtoGENI [1] and PlanetLab [19].

REFERENCES

[1] ProtoGENI. http://www.protogeni.net/trac/protogeni.

[2] E. Blanton, S. Fahmy, and S. Banerjee. Resource management in an active measurement service. In *Proc. of IEEE Global Internet*, 2008.

[3] E. Blanton, S. Fahmy, G. N. Frederickson, and S. Gangam. On the cost of network inference mechanisms. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, April 2011.

[4] P. Calyam, C.-G. Lee, E. Ekici, M. Haffner, and N. Howes. Orchestration of network-wide active measurements for supporting distributed computing applications. *IEEE Trans. Comput.*, 56(12):1629–1642, 2007.

[5] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling file transfers in a distributed network. In *Proc. of ACM PODC*, pages 254–266, 1983.

[6] D. Croce, M. Mellia, and E. Leonardi. The quest for bandwidth estimation techniques for large-scale distributed systems. *SIGMETRICS Perform. Eval. Rev.*, 37(3):20–25, 2009.

[7] C. Dovrolis, P. Ramanathan, and D. Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Trans. Netw.*, 12(6):963–977, 2004.

[8] M. Fraiwan and G. Manimaran. Scheduling algorithms for conducting conflict-free measurements in overlay networks. *Comput. Netw.*, 52(15):2819–2830, 2008.

[9] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000.

[10] J. L. Hintze and R. D. Nelson. Violin Plots: A Box Plot-Density Trace Synergism. *The American Statistician*, 52(2):181–184, 1998.

[11] I. Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981.

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

[13] R. M. Karp, M. Luby, and A. Marchetti-Spaccamela. A probabilistic analysis of multidimensional bin packing problems. In *Proc. of ACM STOC*, pages 289–298, New York, NY, USA, 1984.

[14] H. 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 26–26, 2006.

[15] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and Active Measurement Workshop*, 2003.

[16] D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. ISBN 978-0-387-75968-5.

[17] R. Sherwood, A. Bender, and N. Spring. Discarte: a disjunctive internet cartographer. *SIGCOMM Comput. Commun. Rev.*, 38(4):303–314, 2008.

[18] H. H. Song and P. Yalagandula. Real-time end-to-end network monitoring in large distributed systems. In *Proc. of COMSWARE*, 2007.

[19] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using planetlab for network research: myths, realities, and best practices. *SIGOPS Oper. Syst. Rev.*, 40:17–24, January 2006.

[20] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proc. of ACM IMC*, pages 39–44, 2003.

[21] C. Wu, B. Li, and S. Zhao. Magellan: Charting large-scale peer-to-peer live streaming topologies. In *Proc. of IEEE ICDCS*, pages 62–62, June 2007.

[22] P. Yalagandula, P. Sharma, S. Banerjee, S. Basu, and S.-J. Lee. S3: a scalable sensing service for monitoring large networked systems. In *Proc. of INM*, pages 71–76, 2006.