# On the Schedulability of Measurement Conflict in Overlay Networks

Mohammad Fraiwan and G. Manimaran

Real Time Computing & Networking Laboratory
Dept. of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011
mfraiwan,gmani@iastate.edu

**Abstract.** Network monitoring is essential to the correct and efficient operation of overlay networks, and active measurement is a key design problem in network monitoring. Unfortunately, almost all active probing algorithms ignore the measurement conflict problem: Active measurements conflict with each other - due to the nature of these measurements, the associated overhead, and the network topology - which results in reporting incorrect results. In this paper, we consider the problem of scheduling periodic QoS measurement tasks in overlay networks. We first show that this problem is NP-complete, and then propose a conflict-aware scheduling algorithm whose goal is to maximize the number of measurement tasks that can run concurrently, based on a well known approximation algorithm. Simulation results show that our algorithm achieves 25% better schedulability over the existing algorithm. Finally, we discuss various practical considerations, and identify several interesting research problems in this context.

## 1 Introduction

Overlay networks have come to play an increasingly diverse role in today's network applications. Such applications include end-system multicast, routing, storage and lookup systems (e.g., Akamai [1]), and security. Monitoring overlays are also becoming more common, Internet service providers are deploying monitoring tools at specific nodes in their networks, as part of a Network Measurement Infrastructure (NMI) [2]. Overlays allow designer to implement and deploy their algorithms, applications, and services with great flexibility and versatility. However, maintaining the efficient and correct operation of these overlays requires regular probing of overlay links to measure available bandwidth, delay, and loss rate. Network monitoring is an important infrastructure service that helps in enforcing network security policies and track network performance.

Network monitoring and active measurements in particular do not come at a cheap cost. Measuring bandwidth, loss, and delay involves injecting a non-negligible amount of probe packets [3]. An inherent property of overlay networks is the overlap or correlation between seemingly independent overlay links. Even if the measurement is done at the physical IP-level network, paths between different

source-destination pairs typically overlap [4]. This overlap, in association with the injected overhead, causes concurrent measurements to cross talk or conflict with each other, which results in reporting incorrect results, and hence taking incorrect actions by the network administration or the beneficiary applications [2]. In this paper we formulate the measurement conflict problem as a scheduling problem of periodic QoS real-time tasks. We show the NP-Completeness of this problem, and we propose a heuristic algorithm based on graph partitioning concepts.

The remainder of this paper proceeds as follows. Section 2 gives a background into the measurement conflict problem. Section 3 presents our network monitoring model, its assumptions, and its computational complexity. Section 4 discusses the related work and motivation. Section 5 goes into the details of the algorithms. Section 6 presents simulation results that demonstrate the performance of the technique proposed. We conclude in section 7.

## 2 Background

The measured parameters of a network path are bandwidth, loss rate, latency, jitter, etc. We classify the way these parameters can be obtained into two categories, based on the introduction of traffic into the network. Passive measurement techniques capture and analyze the network traffic traces without injecting any major extra overhead. They are non-intrusive, but may fail in reporting accurate measurements for some scenarios [5]. On the other hand, active measurement techniques are inherently intrusive, since they inject a non-negligible amount of packets into the network [3].

The particular measurement algorithms are not our concern in this paper. However, understanding the way active measurements work, and how they are affected by each other is essential. Active measurement tools start by injecting packets into the network, then measuring the transmission rate of the packet sequence [6], or the changes in the probing packet gaps [7]. Some tools use uniformly spaced packet sequences, while other use statistically calculated intervals between packet trains. Thus, the extra traffic injected by other conflicting tools may change the available bandwidth of the measured path, or cause congestion on shared links that leads to instantaneous spikes in loss rate. Another important issue is the measurement interval, the averaging interval may be short-term, or over a relatively longer period [7]. Tools with longer averaging period tend to have more tolerance toward small amounts of conflict.

Several factors contribute to the measurement conflict problem as follows:

- Overlay and IP-level topologies: Several studies [3] have shown that there is a great deal of overlap at the IP-level between seemingly independent overlay paths. This overlap may cause measurements to interfere with each other depending on the properties of the measurement tools.
- Tasks properties: Tasks can be generally divided as either communication intensive, or computation intensive. Communication intensive tasks inject a non-negligible amount of traffic into the network, so multiple concurrent

tasks, crossing the same path or part of the path, may cause a drop in the reported available bandwidth. On the other hand, computation intensive tasks conflict with other tasks if they cross a common node.
– Administrative constraints usually limit the amount of monitoring traffic that can be injected into the network at any time.

## 3 Network Monitoring Model and Complexity

### 3.1 Network Monitoring Model

We generalize the network model to include overlay networks, as well as IP-level networks. As we said earlier, network monitoring is considered part of a NMI (Network Measurements Infrastructure) employed by ISP's (Internet Service Providers) at the IP-level network. While the rise of overlay networks and their varying application requires regular monitoring to achieve efficient and correct operation of these overlays.

Given an overlay network undirected graph $G_o = (V_o, E_o)$, where $V_o$ is the set of overlay nodes and $E_o$ is the set of overlay edges. The corresponding IP-level graph $G_{IP} = (V_{IP}, E_{IP})$ is also available, where $V_{IP}$ is the set of physical nodes and $E_{IP}$ is the set of physical edges. Note that $V_o \subseteq V_{IP}$, but $E_o \nsubseteq E_{IP}$ in general. Several previous studies have assumed and justified the knowledge of the underlying IP topology by the overlay network operator [3].

Let $T = \{T_1, \ldots, T_n\}$ the set of tasks to be scheduled. $T_i = (s_i, d_i, c_i, p_i, tool_i)$, where $s_i$ is the source of the measurement task, $d_i$ is destination of the same task, $c_i$ is the running time of the task, $p_i$ is the period of task $T_i$, and $tool_i$ is the tool used by the task. The deadline of the task is the same as its period. We also define matrix $M$ to be an $n \times n$ 0-1 matrix, representing the possible conflict between tasks if run at the same time, where $m_{ij} = 1$ if task $i$ conflicts with task $j$, and 0 otherwise. The conflict matrix $M$ captures the conflict among tasks based on the set of tasks and the previously mentioned conflict factors.

### 3.2 Problem Definition

We start by defining three important terms related to this problem:

– Feasibility: A feasible schedule is a schedule in which tasks meet their deadlines and no two tasks are scheduled in a conflicting manner.
– Optimality: A scheduling algorithm is said to be optimal if no other algorithm can find a feasible schedule for a task set that this algorithm failed to find a feasible schedule.
– Schedulability: This defines the efficiency of the scheduling algorithm in terms of the ability to find a feasible schedule.

The measurement conflict scheduling problem can be defined as follows: Given the set of tasks $T$ and the conflict matrix $M$, find a feasible schedule of tasks that maximizes schedulability. We prove that this problem is NP-complete by reduction from Maximum Cardinality Independent Set [8] as follows.

Maximum Cardinality Independent Set is a known NP-complete problem.
INSTANCE: A Graph $H = (W, F)$, where $W$ is the set of vertices and $F$ is the set of edges.
Question: Is there a subset $W' \subseteq W$ such that, for all $u, v \in W'$, $(u, v) \notin F$ and $W'$ is of maximum cardinality? An independent set is said to be of maximum cardinality if it contains the largest possible number of vertices without destroying the independence property.

**Theorem 1.** *The optimal scheduling of measurement tasks is NP-Complete.*

*Proof:* We consider an instance of the problem, where all the tasks have the same period and the same execution time. An optimal scheduling policy will allow for the maximum concurrency of tasks, without violating the conflict constraint. Construct a task conflict graph $G = (V, E)$ as follows. Assign a node for each task in the set of tasks $T$, thus $|T| = |V|$. An edge is added between nodes $i, j \in V$ if the corresponding entry in the conflict matrix M, $m_{ij} = 1$.

Notice that finding a maximum independent set in the conflict graph will correspond to finding a maximum set of non-conflicting tasks that can execute at the same time. Finding the rest of the schedule can be done by repeatedly deleting the nodes in the previous maximum set and their incident edges, and finding the maximum independent set on the new graph. A polynomial time verification algorithm can be easily found, thus the problem is NP-Complete.

Although this problem sounds similar to the well known problem of offline single processor scheduling of real-time tasks, a major difference exists. The resource under consideration is the network on which probe conflicts occur. If the network is treated as a single processor system, then there is no parallelism in executing the tasks, this will lead to no conflicts, but poor schedulability. The problem is also different from the multiprocessor scheduling, because if the network is treated as a multiprocessor system, then the problem become identifying the processors that are active (i.e., the tasks that can execute concurrently) at any given time.

## 4 Related work

Most of the research on network monitoring has focused on reducing the probing overhead [3]. Nonetheless, this reduction is a function of the number of nodes in the probing node set, with the best known algorithms having complexity of $O(n \log n)$. However, the constant factor in this function is high and depends on the type of measurements (e.g., bandwidth, loss, etc.) and other factors.

The measurement conflict problem was first introduced by Calyam et al. in [2]. They have observed such a problem while designing ActiveMon for the Third Frontier Network (TFN) project. ActiveMon is an NMI software Framework to collect and analyze network-wide active measurements. In another study [9] they have developed a simple scripting language interface to specify various measurement requirements used in generating measurement timetables. Our work is different from theirs in that we show the NP completeness of the problem and

provide a far superior alternative algorithm at no extra cost, while providing a performance bound.

A token passing protocol has been used by related studies to minimize collisions between probes [10], this protocol is used to generate time series of measurement data, which is then used in numerical forecasting models as part of a Network Weather Service. However, their approach does not allow for concurrent execution of multiple measurement tasks.

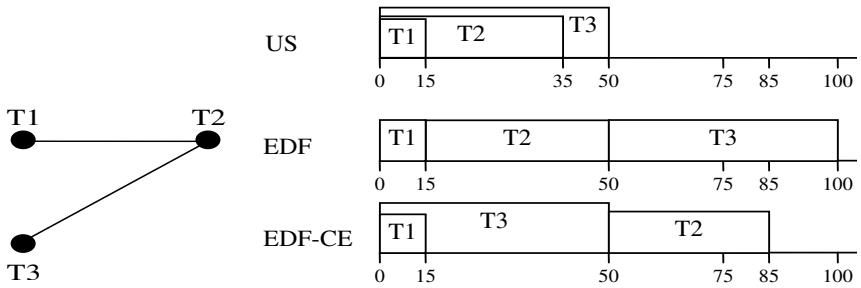
Periodic task scheduling is a well studied problem in the Real-Time scheduling literature. For example, EDF (Earliest Deadline First) scheduling is an optimal single processor scheduling algorithm. As the name suggests, EDF scheduling gives higher priority to the task with the earliest deadline. In this study, we leverage some of the concepts used in Real-Time scheduling, without affecting the novelty of our approach.

### Motivation

In this section we give a motivational example, where we show the shortcomings of existing solutions and the existence of a significant room for improvements. The algorithms to be considered are:

- Unsynchronized scheduling (US): Tasks are run without regard to conflict. This algorithm achieves maximum schedulability, but with a lot of conflicts.
- Non-preemptive EDF: This algorithm will schedule tasks based on deadline, with higher priority given to earlier deadlines. It permits no conflicts, but achieves worst schedulability.
- EDF-CE: EDF with Concurrent Execution, this is the algorithm proposed in [2]. Tasks are executed in EDF order, and ready tasks are added randomly as long as they do not conflict with the currently executing tasks.

**Example 1.** Consider the following three tasks, where $T_i = (c_i, p_i)$. $T_1 = (15, 50)$, $T_2 = (35, 75)$, $T_3 = (50, 100)$. The conflict graph is given in Fig. 1a. This example shows an interesting fact. *When faced with a non-uniform task set, blindly trying to increase task parallelism may lead to a scheduling anomaly, a situation in which a higher priority task misses its deadline due to a lower priority task execution.* The schedule produced by EDF-CE exhibits this problem, see Fig. 1b. Task T1 has the highest priority (i.e., lowest deadline), to achieve maximum overlap, task T3 was allowed to run concurrently with task T1, but since T3 has a longer execution time, it will continue running past T1, causing T2 to miss its deadline. The schedule produced by Unsynchronized Scheduling (US) allows all the tasks to be run at the same time. US and EDF are useless for this problem due to large conflicts and poor schedulability respectively. On the other hand, EDF-CE provides higher schedulability with fewer conflicts, but it does not provide any guarantees on the amount of parallelism, misses many chances for improvement by randomly choosing tasks for parallel execution, and may cause scheduling anomalies.




(a) Task Conflict Graph (b) The schedules produced by US, EDF, and EDF-CE.

Fig. 1: Example 1. Under EDF the second instance of $T_1$ will miss its deadline, while under EDF-CE, $T_2$ will miss its deadline due to a scheduling anomaly, and the algorithms abort.

## 5 The Scheduling Algorithm

Since the problem is NP-complete, we develop a heuristic algorithm based on graph partitioning. The algorithm generally has the following three steps:

1. Construct the task conflict graph.
2. Partition the task conflict graph into the *least number of partitions.*
3. Schedule each partition concurrently as long as there is enough slots in the time frame, where the time frame is the period of the uniform task set (*i.e.,* task set with the same period and deadline), or the least common multiple (LCM) of the periods in a non-uniform task set.

### 5.1 The Conflict-aware Scheduling Algorithm

In case the task set is uniform, the problem becomes a partitioning problem. As for the general case of a non-uniform task set, we use period transformation and execution time transformation [11]. Tasks within each partition are transformed into a certain number of the same uniform task $\tau = (C, P)$. This uniform task can be the same across partitions or different. Transforming the tasks into a common task ensures that all the tasks are aligned with each other. Thus, achieving higher parallelism, while maintaining the properties of the original tasks (*i.e.,* deadline, and utilization). Algorithm 1 shows the pseudo code for the conflict-aware scheduling algorithm, which proceeds in the following steps:

1. The current task set is grouped into a collection of non-uniform task partitions (steps 1.3-1.7), using a least conflict first partitioning algorithm. The task with the least number of conflicts is chosen first. Then, all of its neighboring (*i.e.,* conflicting) tasks in the graph are removed. We proceed until no more tasks can be added to the current set (step 1.4). The tasks from

**input** : The Task Conflict Graph $G = (V, E)$.
$\tau = (C, P)$
**output**: A schedule of tasks

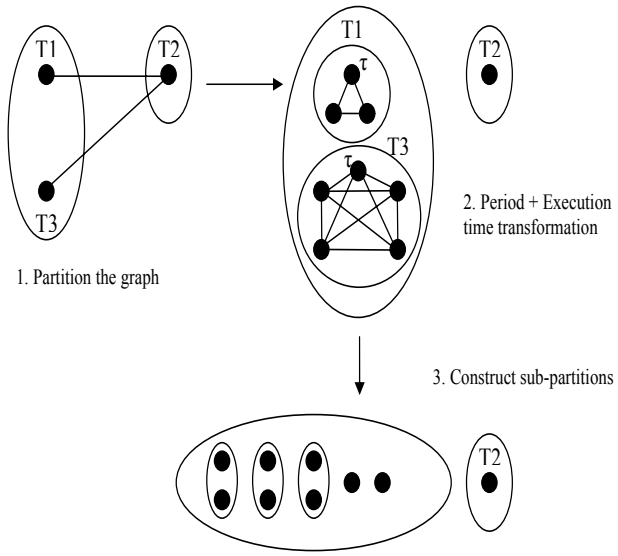
1.1 $S \leftarrow \phi$
1.2 $I \leftarrow \phi$
1.3 **while** $G \neq \phi$ **do**
1.4 $\quad I \leftarrow Partition(G)$
1.5 $\quad G \leftarrow G - I$
1.6 $\quad S \leftarrow S \cup I$
1.7 **end**
1.8 **foreach** *subset* $s \in S$ **do**
1.9 $\quad$ *Transform task subsets using* $\tau$
1.10 $\quad$ *Construct a subset conflict graph* $H$*, where subtasks from the same parent task share an edge*
1.11 $\quad$ *Construct sub-partitions from* $H$
1.12 **end**
1.13 *The earliest deadline of a parent task in a sub-partition is the deadline of that sub-partition.*
1.14 *Sort all the resulting sub-partitions in increasing order of deadline. Output the schedule.*

**Algorithm 1**: The conflict-aware scheduling algorithm

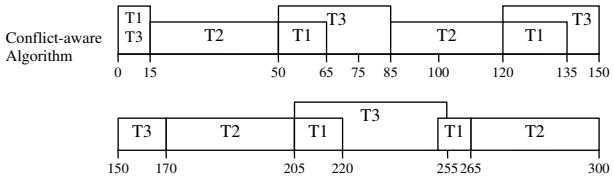
the resulting partition are removed from the graph, along with their incident edges (step 1.5). The process is repeated until all tasks are grouped.

2. Within each partition, tasks are divided into multiple uniform tasks $\tau = (C, P)$, where P is the common period and it is smaller than the period of any task in the task set, C is the common execution time and it is smaller than the execution time of any task in the task set, and the utilization $\frac{C}{P}$ is smaller than the utilization of any other task. Subtasks from the same parent task will form a complete subgraph in the task conflict graph (i.e., they pairwise conflict), and each subtask will inherit the conflicts of its parent task. The number of subtasks generated from a given tasks is given by: $\lceil \frac{c_i/p_i}{C/P} \rceil$
There may be some performance loss due to the rounding up. We argue, however, that achieving higher parallelism, and no scheduling anomalies overrides this slight loss of performance. Note that $\tau$ can be the same across all partitions, or unique to each partition.
*Each subtask will inherit the deadline of its parent task. Hence, inherit the priority. So the deadline of the subtask is not* $P$*, which is the period of* $\tau$*.*
3. Each partition is further divided into sub-partitions, using the same partitioning algorithm (step 1.11).
4. The sub-partitions from all partitions are in increasing order of the earliest deadline of a parent task in the sub-partition. This is the schedule.

Going back to example 1, we use $\tau = (5, 50)$ to normalize the tasks. Task $T_1$ is split into 3 sub-tasks, while $T_3$ is split into 5 subtasks, and $T_2$ stays the same. This division is useful for the purpose of finding maximum overlap, while




(a) The working of algorithm 1.




(b) The schedule produced by algorithm 1.

Fig. 2: Algorithm 1 solution to example 1.

preserving task priorities. However, the tasks actual deadline is still the same. For example, in figure 2 at time 50, 3 sub-tasks of $T_3$ will miss their deadline imposed by $\tau$, but the actual deadline is 100.

### 5.2 Implementation Issues

We envision this scheduling algorithm as a component of a monitoring infrastructure, which is part of a larger network management system, or a network application. A node is selected as a controller, which is responsible for collecting and scheduling measurement requests. To make the controller fault tolerant, well-known backup and leader election strategies can be used. As for the performance bottleneck concern, we argue that the communication and computation costs at the central node are small for a reasonably sized monitoring task set.

Network measurement is a sophisticated process that involves many issues that need to be considered in conjunction with scheduling. First, the measurement tool may not be susceptible to subdivision. For example, bandwidth measurement tools use the dispersion of packets in packet trains to estimate the bandwidth, the designer need to be careful not to subdivide the packets in the same train as it will affect the measurement process. Second, the execution time of measurement tools may vary greatly based on spatial and temporal factors such as hop count and path bandwidth. And for some measurement tools (e.g., PathChirp [12]) the difference may be in the order of minutes. Predetermining each task's run-time and period may not be a trivial issue.

Period and execution time transformation may lead to increased execution time of the scheduling algorithm. Since the scheduling algorithm is either run offline or run only when changes in the task set occur, this kind of increase is moderate considering that the least conflict first partitioning algorithm runs in linear time [13], and the processing capabilities of the centralized controller.

Another implementation issue is that of the global synchronization of measurement schedules starts and stops. Achieving a 100% synchronization of a distributed system clock is a difficult task. However, our studies, omitted here for the lack of space, show that complete synchronization is not necessary as the measurements accuracy gracefully degrades with the amount of overlap. Thus, a certain amount of synchronization imperfection can be tolerated.

## 6 Simulation Results

Scheduling uniform task sets depends solely on how good the partitioning algorithm performs. To study this effect, we experiment with a task conflict graph consisting of 100 tasks, each pair of tasks share an edge based on a certain conflict probability. We vary the conflict probability from 0.1 to 0.9 , and we report the average number of partitions generated over 20 runs. Figure 3 shows that the proposed least conflicts first approach achieves about *10% less number of partitions than EDF-CE.*

We now examine the schedulability of our conflict-aware algorithm. In particular, we study the success ratio of our approach in comparison to existing solutions. The success ratio is defined as:

$$success\ ratio = \frac{\#\ of\ tasks\ successfully\ scheduled}{Total\ number\ of\ tasks}$$

We used a set of 20 tasks, and took the average of 10 runs. The period of each task is selected uniformly at random from $[100, 1000]$, we report the success rate as a function of the conflict probability among tasks for task utilization values of 0.2, 0.4, 0.6. The execution time of each task is the product of its period and the utilization value used in the corresponding figure.

Figure 4 shows that the conflict-aware scheduling algorithm tasks achieves up to 25% better success ratio. The main reasons for this improvement are the elimination of scheduling anomalies, higher parallelism achieved by our algorithm,

and a higher density of tasks in each partition (*i.e.,* a small number of partition have a large number of tasks), which results in dropping low density partitions. In addition, the figure shows that our conflict-aware approach and the EDF-CE algorithm follow the same trend as the conflict probability increases. At low conflict probabilities (*i.e,* below 0.2), there is a great deal of possible tasks concurrency, and both algorithms achieve high schedulability. In the region between 0.2 and 0.8 conflict probabilities, the conflict-aware algorithm achieves superior schedulability compared to EDF-CE. This superiority decreases with increasing task utilization due to the smaller room for improvements. Both algorithm converge again at high conflict probabilities (*i.e.,* probabilities greater than 0.9), where most of the tasks conflict with each other, and they must be executed sequentially to avoid conflicts.

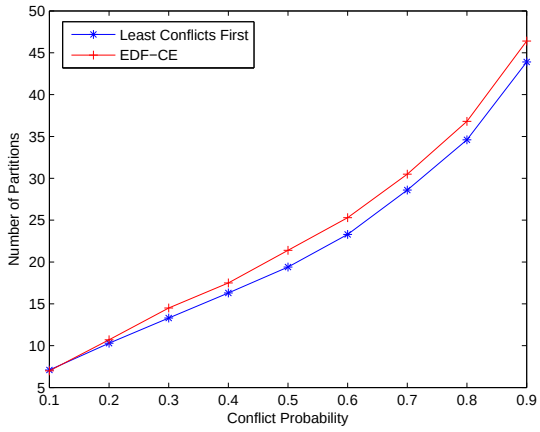
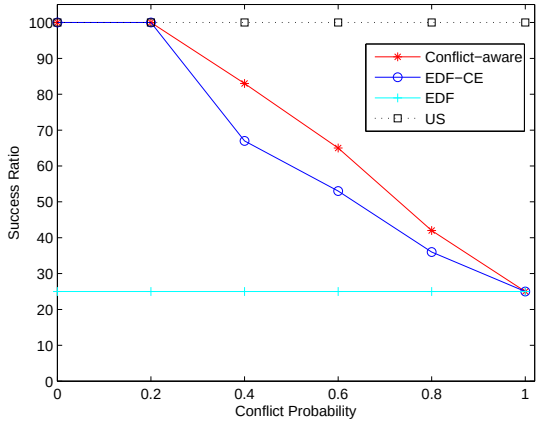


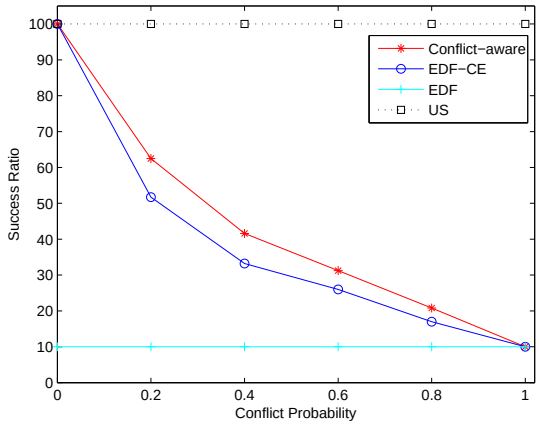

Fig. 3: The number of partitions produced by the EDF-CE algorithm, and the least conflicts first partitioning algorithm.

## 7 Conclusion and Future Work

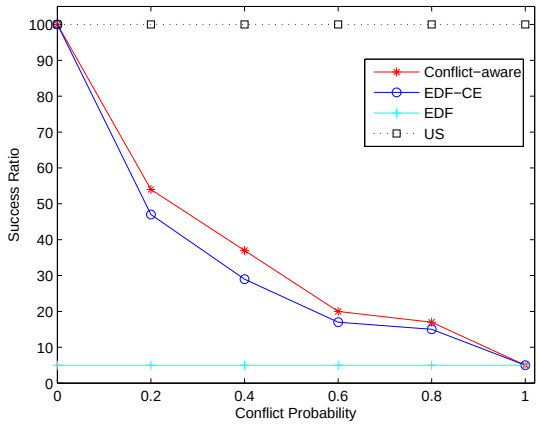
In this paper, we have shown that the problem of optimally scheduling measurement tasks is NP-complete. We proposed a polynomial time heuristic scheduling algorithms based on a well-known graph theory approximation algorithm, which achieves 10% less number of partitions. Simulation studies have shown that our algorithm improves schedulability by 25% compared to existing solutions. Future work includes conducting intensive performance evaluation of the proposed algorithm (e.g., bounding the performance deviation from the optimal), and performing experiments on PlanetLab [14]. In addition, we plan to formulate imprecise measurement scheduling problems where tasks are allowed to partially overlap, and methodically developing solutions for them.




(a) Task utilization = 0.2




(b) Task utilization = 0.4




(c) Task utilization = 0.6

Fig. 4: Success ratio as a function of conflict probability.

## References


1. Akamai Technologies (February 14, 2007); http://www.akamai.com
2. Calyam, P., Lee, C., Arava, P., Krymskiy, D.: Enhanced EDF scheduling algorithms for orchestrating network-wide active measurements. Proc. of IEEE RTSS'05.
3. Tang, C., Mckinley, P.: On the cost-quality tradeoff in topology-aware overlay path probing. Proc. of IEEE ICNP 2003.
4. Cui, W., Stoica, I., Katz, R.: Backup path allocation based on a correlated link failure probability model in overlay networks. Proc. of IEEE ICNP 2002.
5. Zseby, T., Zander, S., Carle, G.: Evaluation of Building Blocks for Passive One-Way-Delay Measurements. Passive and Active Measurements Workshop 2001.
6. Jain, M., Dovrolis, C.: End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. Proc. of SIGCOMM'02.
7. Strauss, J., Katabi, D., Kaashoek, F.: A measurement study of available bandwidth estimation tools. Proc. of ACM IMC Oct. 2003.
8. Garey, M., Johnson, D.: Computers and Intractability: A guide to the theorey of NP-completeness. W. H. Freeman, 1979.
9. Calyam, P., Lee, C., Arava, P., Krymskiy, D., Lee, D.: OnTimeMeasure: A Scalable Framework for scheduling active measurements. Proc. IEEE E2EMON 2005.
10. Gaildioz, B., Wolski, R., Tourancheau, B.: Synchronizing Network Probes to avoid Measurement Intrusiveness with the Network Weather Service. IEEE High-performance Distributed Computing Conference 2000.
11. Sha, L., Lehoczky, J., Rajkumar, R.: Solutions for some practical problems in prioritized preemptive scheduling. Proc. 7th IEEE RTSS 1986.
12. Ribeiro, V., Riedi, R., Baraniuk, R., Navratil, J., Cotrell, L.: pathchirp: Effecient available bandwidth estimation for network paths. In Passive and Active Measurement Workshop 2003.
13. Halldorsson, M., Radhakrishnan, J.: Greed is good: approximating independent sets in sparse and bounded degree graphs. Proc. of the twenty-sixth annual ACM symposium on Theory of computing. 1994.
14. PlanetLab (February 14, 2007); http://www.planet-lab.org/