Orchestration of Network-wide Active Measurements for Supporting Distributed Computing Applications

Prasad Calyam, *Student Member, IEEE,* Chang-Gun Lee, *Member, IEEE,* Eylem Ekici, *Member, IEEE,* Mark Haffner, *Student Member, IEEE,* and Nathan Howes, *Student Member, IEEE*

***Abstract*— Recent computing applications such as videoconferencing and Grid computing run their tasks on distributed computing resources connected through networks. For such applications, knowledge of the network status such as delay, jitter, and available bandwidth can help them select proper network resources to meet the Quality of Service (QoS) requirements. Also, the applications can dynamically change the resource selection if the current selection is found to experience poor performance. For such purposes, Internet Service Providers (ISPs) have started to instrument their networks with Network Measurement Infrastructures (NMIs) that run active measurement tasks periodically and/or on-demand. However, one problem that most network engineers have overlooked is the *measurement conflict problem*, which happens when multiple active measurement tasks inject probing packets to the same network segment at the same time, resulting in misleading reports of network performance due to their combined effects. This paper proposes enhanced EDF (Earliest Deadline First) algorithms that allow "Concurrent Executions" to orchestrate offline/online measurement jobs in a conflict-free manner. The simulation study shows that our measurement scheduling mechanism can improve the schedulable utilization of offline measurement tasks up to 300% and the response time of on-demand jobs up to 50%. Further, we implement and deploy our scheduling mechanism in a real working NMI for monitoring the Internet2 Abilene network. As a case study, we show the utility of our algorithms in the widely-used Network Weather Service (NWS).**

***Index Terms*— Active network probes, Measurement conflict, Real-time scheduling, Concurrent execution, Network Weather Service.**

I. Introduction

Recent computing applications such as videoconferencing and Grid computing utilize distributed computing resources connected through the Internet. Thus, their user-level performance relies on the status of the Internet paths they use[1]. If we can measure and predict the status, we can select the computing resources and their connecting Internet paths that can soft guarantee the user-level QoS.

A preliminary version of this paper has appeared in the proceedings of IEEE Real-Time Systems Symposium (RTSS), 2005 [1].

Manuscript received July 12, 2006; revised February 02, 2007.

P. Calyam is with OARnet and The Ohio State University, Columbus, OH 43210, USA, Email: pcalyam@oar.net; C. -G. Lee, the corresponding author, is with the Seoul National University, Seoul, 151-742, Korea, Email: cglee@snu.ac.kr; E. Ekici, M. Haffner and N. Howes are with The Ohio State University, Columbus, OH 43210, USA, Email: ekici@ece.osu.edu, {haffner.12,howes.16}@osu.edu

[1]There are several ways to map the network measurements to the user-level quality. The E-Model [2], for example, is a standardized computational model by ITU-T to estimate the user-level VoIP quality from the measured network status.

Therefore, for the success of distributed computing applications, it is critical to collect Internet status measurements in an accurate and timely manner. Fortunately, Internet Service Providers (ISPs) have started to instrument their networks with Network Measurement Infrastructures (NMIs) [3], [4], [5] for continuous monitoring and estimation of network-wide status. For this, they use active measurement tools such as Ping, Traceroute, H.323 Beacon [2], Iperf [6], Pathchar [7] and Pathload [8] that actively inject probing packets to collect useful measurements such as end-to-end delay, jitter, loss, bandwidth, etc. The NMIs periodically run these measurement tools on the measurement servers at strategic points to collect the periodic sampling of network status, which is essential for network status prediction [9]. They also can run the measurement tools on-demand for applications that require a more detailed look about certain network paths.

When executing the periodic and on-demand measurement jobs, an important problem we recently observed is the *measurement conflict problem*, which has been overlooked by most network engineers. Since active measurement tools consume non-negligible amount of network resources for injecting probing packets, if two or more measurement jobs run concurrently over the same path, they can interfere with each other resulting in misleading reports of network status. Our experiment in Figure 1 illustrates the measurement conflict problem. In the experiment, we connect two measurement servers by a LAN Testbed with 1500 Kbps bandwidth and run one H.323 videoconferencing session at 768 Kbps dialing speed as the background traffic. Thus, the remaining bandwidth should be approximately 732 Kbps. Given that streaming media and videoconferencing traffic is essentially UDP traffic, Iperf in UDP mode is popularly used to measure the available bandwidth. Thus, we make the two servers occasionally initiate Iperf jobs to monitor the available bandwidth. When we make two Iperf jobs run back-to-back with mutual exclusion (shown in the left-half of Figure 1), their measurements are in agreement with our expectation. However, when we intentionally make two Iperf execution durations overlap (shown in the right-half of Figure 1), it causes misrepresentation of the remaining bandwidth, merely due to conflicts of two Iperf jobs. This implies that if measurement tools are initiated without being orchestrated with each other, their execution duration may overlap resulting in misleading measurement reports.

This observation motivates a *scheduling problem of measurement jobs for orchestrating them to prevent conflicts while still providing the periodicity of periodic measurement jobs and quick response to the on-demand measurement jobs.* The nature of the problem is similar to real-time scheduling even though the time granularity of periods is much coarser (order of minutes) than that

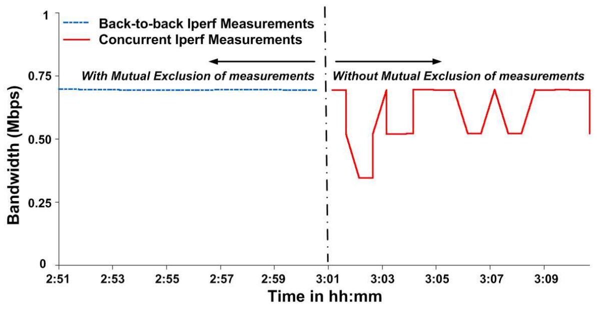


Fig. 1. Iperf test results with and without mutual exclusion of measurements

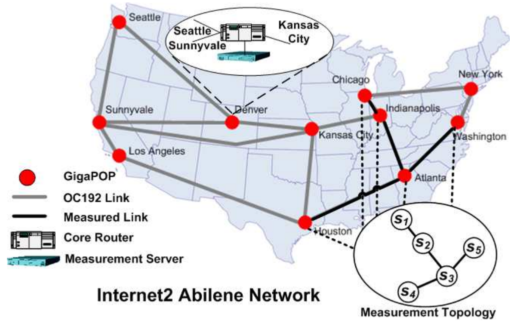


Fig. 2. Measurement Topology constructed for a set of network paths

of the classical real-time systems. The measurement scheduling problem, however, has a fundamental difference from the classical real-time scheduling problems: More than one measurement job can be scheduled at the same time on the same server and the same network path as long as they can produce the correct measurement data. We call this a "concurrent execution" of multiple jobs with "no conflict".

This paper proposes conflict-free scheduling algorithms for measurement jobs leveraging the real-time scheduling principles and the concurrent execution. More specifically, the contributions of this paper can be summarized as follows:

- We propose an offline scheduling algorithm based on the EDF principle [10] but allowing concurrent execution if possible, which can significantly improve the schedulability of a given set of periodic measurement tasks,
- We propose an online mechanism that can steal left-over times from the offline schedule to serve on-demand measurement requests as early as possible without violating the periodicity requirements of existing measurement tasks,
- We implement an actual NMI scheduling framework equipped with the proposed scheduling mechanisms to measure an operational network, Internet2 Abilene network.

The rest of this paper is organized as follows: The next section summarizes the related work. Section III formally defines the measurement task scheduling problem. Section IV presents our offline and online measurement scheduling algorithms. Section V presents our case study for applying the proposed scheduling algorithms to the Network Weather Service (NWS) and its use for distributed computing. In Section VI, our experimental results from both simulations and actual implementation are presented. Finally, Section VII concludes the paper.

II. Related Work

Many of the earliest Network Measurement Infrastructures (NMIs) used simple *Ping* and *Traceroute* measurements without paying attention to possible overlaps of their execution durations. This is acceptable since they are neither CPU nor channel intensive, allowing overlaps without causing measurement conflicts. However, many of today's NMIs such as NLANR AMP [3], Internet2 E2EpiPES [5], NWS [9], Surveyor [11] and RIPE [12], employ toolkits that have several CPU and/or channel intensive measurement tools, which may cause measurement conflict problems. Nevertheless, these NMIs use a simple scheme that creates *cron* jobs that start active measurements at the planned periodic time points without paying attention to avoiding measurement conflicts. As a result, they can give erroneous measurement results and fail to reflect the actual network status.

To address the measurement scheduling problem, [3] and [11] use a simple round-robin approach where measurement servers take turns such that only one tool executes at a time. In NMIs such as [5], a resource broker scheduling scheme is used. Using this resource broker scheme, multiple measurement requests are queued for scheduling and executed on a first-come first-serve basis on a measurement server. The network weather service (NWS) uses a token-passing mechanism [13] in an attempt to meet the measurement periodicity requirements while obtaining accurate network status information. This mechanism allows only a single server in possession of a token to initiate measurements. The round-robin, resource broker, and token-passing are similar in principle, i.e., they allow only one instance of measurement to be executed at a time. Therefore, they cannot leverage the concurrent execution of multiple measurement jobs and hence limit the schedulability.

In addition to our contributions from the scheduling perspective, another significant contribution is our systematic scheduling framework that automates the whole process from the measurement specification to the runtime measurement data collection. None of the previous schemes provide such a systematic framework. As a result, existing schemes require considerable time and effort to specify distinct sampling requirements, add or delete measurement tasks, and generate measurement schedules accordingly. Furthermore, it is hard to implement the policy contracts among multiple ISPs for measurements across ISP borders. With our systematic measurement framework, however, the entire process can be automated and the manual effort minimized.

III. Problem Description and Terminology

An ISP deploys measurement servers at strategic points to continuously estimate the network-wide status. The measurement servers measure the network paths to other servers. They are attached to core routers as shown in the case of the Denver core router in Figure 2. The paths to be measured are specified by a measurement topology, which can be formally represented by a graph $G = (N, E)$, where N is the set of measurement servers and E is the set of edges between a pair of servers. Figure 2 shows an example measurement topology that consists of measurement servers $N = \{S_1, S_2, S_3, S_4, S_5\}$ and edges among S_1, S_2, S_3, S_4, and S_5.

On top of the measurement topology, a set of periodic measurement tasks is specified. Each periodic measurement task τ_i is specified to measure a path from a source server src_i to a destination server dst_i using an active measurement tool $tool_i$. The measurement should be periodically repeated with period p_i. The j-th instance (or *job*) of τ_i is denoted by τ_{ij}. The time when the j-th job τ_{ij} is released is called the *release time* and simply given by $(j-1)p_i$. The execution time of a single measurement instance is denoted by e_i. Then, a periodic measurement task can be represented using the similar notion of a real-time periodic task as follows:

$$\tau_i = (src_i, dst_i, tool_i, p_i, e_i).$$

The set of all offline specified periodic measurement tasks is denoted by

$$\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}.$$

We also define a *hyperperiod* for task set Γ as the least common multiple of all the task periods in the set. Thus, each hyperperiod repeats the same pattern of release times. Therefore, the same schedule constructed for a single hyperperiod can be used repeatedly.

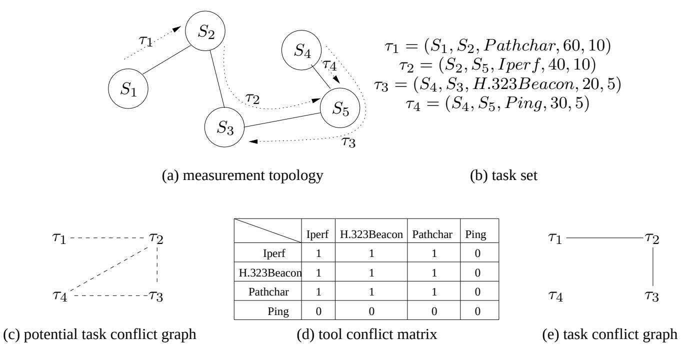
In addition to such offline specified measurement tasks, there can be on-demand measurement requests to quickly collect customized measurements. For example, an Internet network engineer might want to trace-back the source of a DoS attack as soon as possible by running on-demand measurement jobs over suspicious paths [14]. Such an on-demand measurement request is denoted by

$$J_k = (src_k, dst_k, tool_k, e_k).$$

For such an on-demand request, a quick response is desirable. Thus, as a performance metric, we use the *response time*, which is defined as the time difference between the time when the measurement job is requested and the time when the request is finally served.

Our problem is to schedule the aforementioned offline and online measurement jobs on a given measurement topology. Unlike the OS-level schedule that determines when the OS threads and packets can be executed and transmitted, the measurement-level scheduling problem is to determine the start and stop times of a measurement tool whose execution can last a few minutes to have a statistically stable measure. For such measurement-level scheduling, an important constraint is a measurement conflict problem. Overlapping the execution intervals of two measurement jobs may or may not be problematic, depending on the measurement tools used. If a measurement tool is neither CPU intensive nor channel intensive like Ping, it does not interfere with other tools. Thus, overlapping its execution interval with others on the same server and/or path can still give us correct measurement reports. Such an overlap is called a "concurrent execution" with no conflict, which is desirable to improve the schedulability. On the other hand, other active measurement tools such as Iperf [6] and Pathchar [7] are CPU intensive for sophisticated calculations and/or channel intensive due to a large amount of probing packets. Thus, overlapping their execution intervals over the same measurement server or the same channel can cause serious interference and lead to misleading reports of the network status. We define a *measurement conflict* as an execution overlap of multiple measurement jobs that results in misleading reports.

In addition to the measurement conflict issue, one additional constraint of the measurement-level scheduling problem is the *Measurement Level Agreement* (MLA). Utilizing excessive network resources just for active measurements is not appropriate



	Iperf	H.323Beacon	Pathchar	Ping
Iperf	1	1	1	0
H.323Beacon	1	1	1	0
Pathchar	1	1	1	0
Ping	0	0	0	0

Fig. 3. Task conflict graph

since it significantly degrades the regular user traffic performance. Thus, we need a regulation on the measurement traffic. Since an end-to-end measurement could involve analyzing data along network paths of multiple ISPs, we envision "measurement federations" in which many ISPs participate in inter-domain measurements based on MLAs for reaping the mutual benefits of performing end-to-end path measurements. MLAs can specify that only a certain percentage (1% - 5%) or only a certain number of bits per second (1 Mbps - 2 Mbps) of the network bandwidth in ISP backbones could be used for measurement traffic, which can ensure that the actual application traffic is not seriously affected by measurement traffic[2]. We use the notation ψ to denote the MLA specification in an NMI. In the measurement-level scheduling problem, the sum of the bandwidth usage by concurrent measurement jobs over the same channel should be less than ψ at all times.

From the above inputs and constraints, the measurement-level scheduling problem can be formally described as follows:

Problem: Given measurement topology $G = (N, E)$ and offline specified measurement task set $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$, find the schedule of measurement jobs such that all deadlines (equal to periods) can be met while preventing conflicts and adhering to the MLA constraint ψ. For an on-demand measurement request J_k, schedule it as early as possible without violating deadlines of offline tasks in Γ, conflict constraint, and MLA constraint.

IV. Measurement Scheduling Algorithms

In this section, we first present an offline scheduling algorithm to construct a schedule table for a given set of periodic measurement tasks $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$. Then, we present an online algorithm to schedule an on-demand measurement request J_k without missing deadlines of periodic tasks. We first assume existence of a central regulator that governs the global schedule and later relax this assumption.

A. Offline Scheduling Algorithm

In our measurement scheduling framework, a central regulator collects all specifications of periodic measurement tasks and builds a schedule table that determines times when measurement jobs can start and stop at each server. To build such a table,

[2]Since most active measurement tools have options to specify packet sizes and bandwidth usage of a measurement test, simple calculations can be used to determine how much of a network's bandwidth will be used by a given set of active measurements, over a certain period of time.

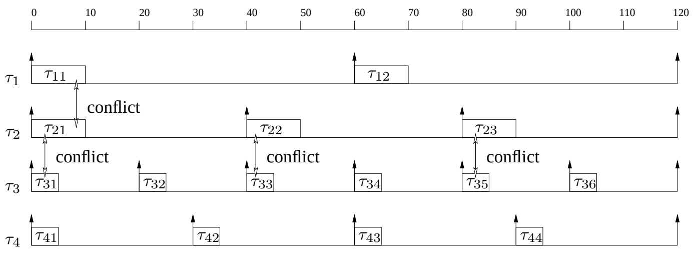


Fig. 4. No orchestrated schedule

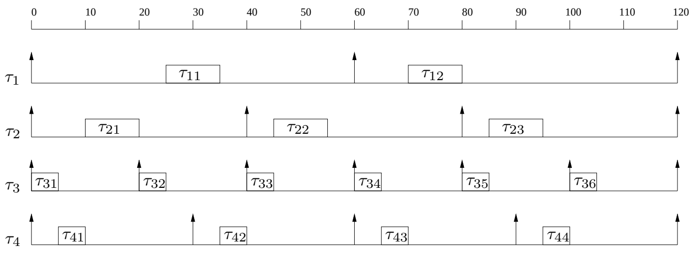


Fig. 5. Orchestration based on single processor non-preemptive EDF schedule

the first step is to make a *task conflict graph* by combining the measurement topology G and the task set Γ. Figure 3 shows an example problem. For the given measurement topology and the task set in Figures 3(a) and (b), we examine each pair of tasks τ_i and τ_j to see if they share the same source server, destination server or part of the paths between source and destination servers. If so, the two tasks may "potentially" conflict if scheduled concurrently. In Figures 3(a) and (b), τ_1 and τ_2 share S_2 and thus we add a potential dependency edge between them in the potential task conflict graph as in Figure 3(c). τ_2 and τ_3 share the path and thus a dependency edge is added. On the other hand, τ_1 does not share any network resource with τ_3 and thus no edge is added. Even if two tasks share network resources, they may not actually conflict depending on the active measurement tools used. Based on our empirical studies in [15], we could determine which tools conflict if they run concurrently. The result is summarized by the tool conflict matrix in Figure 3(d). For example, Iperf and Pathchar conflict if they run concurrently on the same server since both intensively use server and channel resources for active measurement. On the other hand, Ping just injects small probing packets and hence does not conflict with any other tools. Considering the tool conflict matrix, the potential task conflict graph in Figure 3(c) can be converted to the final task conflict graph of Figure 3(e). The edge between two tasks in the task conflict graph means that they should be scheduled in a mutually exclusive manner, otherwise a conflict happens resulting in misleading reports.

Now, we can consider only the final task conflict graph to compute the offline schedule. One obvious solution is to start a measurement job at the source server at its release time without considering measurement conflict and MLA constraints. Figure 4 shows such a schedule for the problem given in Figure 3. In the figure, the upward arrows indicate the release times of the periodic measurement tasks. The schedule, however, causes a number of conflicts that result in misleading report of the actual network performance. Another approach is to run only a single measurement job at any time instant using a non-preemptive EDF scheduling algorithm. Figure 5 shows such a schedule for the same problem. It can completely prevent conflicts. However, it does not allow concurrent execution of multiple jobs even if they do not conflict, which degrades the schedulability.

We aim to find a schedule in between these two extremes such that conflicts are completely prevented while maximizing the concurrent execution whenever possible. For this, we propose the EDF-CE (i.e., EDF with Concurrent Execution) algorithm that schedules measurement jobs in the EDF order while allowing concurrent execution if jobs do not conflict. The algorithm is formally described in the following:

EDF-CE: for the given task conflict graph, find the measurement schedule during a hyperperiod

```
Input: task set $\Gamma$ and task conflict graph
Output: start time $st_{ij}$ and finish time $ft_{ij}$ for each job $\tau_{ij}$ in a hyperperiod
begin procedure
1. Initialize $rt\_list$ with the ordered list of all release times in a hyperperiod
2. Initialize $ft\_list$ = {} /* ordered list of finish times*/
3. Initialize $pending\_job\_queue$ = {}
4. do
5.     $time$ = get the next scheduling time point from $rt\_list$ and $ft\_list$
6.     add all newly released jobs at $time$ to $pending\_job\_queue$ in EDF order
7.     for each job $\tau_{ij}$ in $pending\_job\_queue$ in EDF order
8.         if $\tau_{ij}$ does not conflict with any of already scheduled jobs at $time$ and
9.            scheduling $\tau_{ij}$ at $time$ does not violate MLA constraint $\psi$
10.            $st_{ij} = time$ and $ft_{ij} = time + e_i$
11.            if $ft_{ij}$ is later than the deadline of $\tau_{ij}$
12.                return error /* infeasible task set */
13.            end if
14.            remove $\tau_{ij}$ from $pending\_job\_queue$
15.            add $ft_{ij}$ to $ft\_list$ in order
16.        else
17.            do nothing /* $\tau_{ij}$ will be considered again at the next scheduling time point in the outer loop */
18.        end if
19.    end for
20. until $time == hyperperiod$
end procedure
```

The EDF-CE algorithm maintains the ordered list of release times rt_list and the ordered list of finish times ft_list. Line 1 initializes rt_list with all release times in a hyperperiod. In Figure 6, the release times are 0, 20, 30, 40, 60, 80, 90, 100, and 120. Line 2 initializes ft_list as empty since no job is scheduled yet. Note that the only time points when we need to make a scheduling decision are either when a new job is released or a current executing job is finished. Thus, we call times in rt_list and ft_list "scheduling time points". In addition, the algorithm maintains a $pending\_job\_queue$ that holds all jobs released but not scheduled, in the EDF order. Line 3 initializes it as empty. The **do-until** loop from Line 4 to Line 20 progresses the virtual time variable $time$ upto a hyperperiod while determining the schedule at all scheduling time points. Line 5 moves $time$ to the next scheduling time point. Then, Line 6 adds all newly released jobs

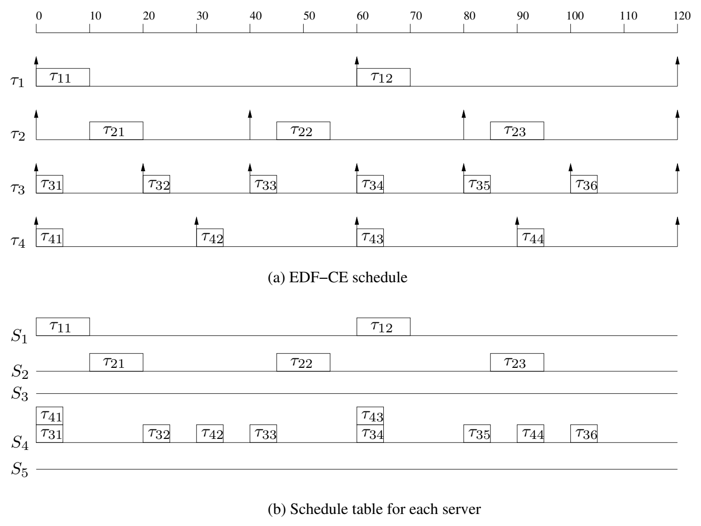


Fig. 6. EDF-CE Schedule

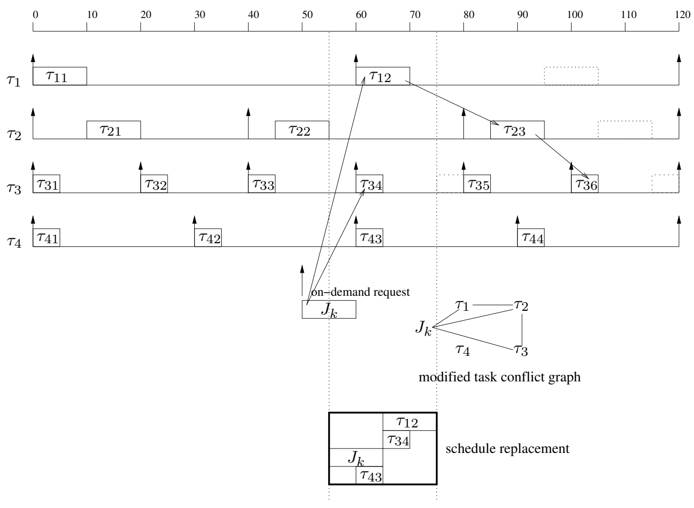


Fig. 7. Recursive pushing for maximum slack calculation

to the $pending\_job\_queue$. The **for** loop from Line 7 to Line 19 examines the pending jobs in the EDF order and determines whether they can start at $time$ without causing any conflict and without violating MLA ψ (see Lines 8 and 9). If so, the job τ_{ij}'s start time st_{ij} is determined as $time$ and its finish time ft_{ij} is determined as $time + e_i$ in Line 10. If the finish time ft_{ij} is later than the deadline of job τ_{ij} in Line 11, we cannot construct a feasible schedule meeting all deadlines and hence return error in Line 12. If we can meet the deadline of τ_{ij}, we can continue. In Line 14, τ_{ij} is removed from the $pending\_job\_queue$. Also, its finish time ft_{ij} is added to ft_list so that ft_{ij} can be considered as a new scheduling time point in the outer **do-until** loop. If τ_{ij} cannot be scheduled at $time$ (Line 16), it is kept in the $pending\_job\_queue$ and can be considered again at the next scheduling time point by the outer loop. Note that the algorithm tries to concurrently start as many jobs as possible in the EDF order at $time$ as long as they neither conflict nor violate the MLA. Figure 6(a) shows such EDF-CE schedule for the same problem of Figure 3. At time 0 of the EDF-CE schedule, note that τ_{11}, τ_{31}, and τ_{41} are executed concurrently but not τ_{41}, which is maximizing the concurrent execution guaranteeing no-conflict.

Once we find the EDF-CE schedule, we can convert it to the measurement schedule table of each server considering the source server of each job. Figure 6(b) shows the schedule tables of all the five servers. Such constructed schedule tables are transferred to corresponding servers so that they can start and stop the planned measurement jobs.

B. Online Scheduling of On-Demand Measurement Requests

At the run time, while each server executes periodic measurement tasks according to the pre-computed schedule table, a network engineer can request an on-demand measurement J_k. For now, we assume that such a request is received by the central regulator.

Upon the arrival of an on-demand request $J_k = (src_k, dst_k, tool_k, e_k)$, our goal is to serve it as early as possible without missing any deadlines of periodic measurement tasks. For this, we propose a *recursive push* algorithm that recursively pushes offline scheduled periodic jobs within their deadlines. This push can create a left-over time called a *slack* as early as possible and this slack time can be used to schedule J_k. The basic idea of recursive push can be best illustrated by Figure 7 that shows the same EDF-CE schedule as above. Suppose that an on-demand request $J_k = (S_2, S_3, \mathit{Iperf}, 10)$ arrives at time 50. We assume that J_k conflicts with τ_1, τ_2, and τ_3 as shown by the modified task graph. The central regulator cannot allow J_k to start at its arrival time 50 since it conflicts with τ_{22}. Thus, the central regulator calculates the maximum slack from starting at 55. For this, the central regulator calls **push**(τ_{12}) and **push**(τ_{34}) to determine how much τ_{12} and τ_{34} can be pushed to make the maximum slack for J_k. The **push** operation is recursive. To determine the maximum **push** of τ_{12}, we first have to know the maximum **push** of the dependent job τ_{23}. Thus, **push**(τ_{12}) recursively calls **push**(τ_{23}) and in turn **push**(τ_{23}) calls **push**(τ_{36}). On the other hand, τ_{36} does not conflict with any other offline scheduled jobs while being pushed up to its deadline $d_{36} = 120$. Such a job with which the recursion can terminate is called a *terminal job*. Similarly, τ_{34} is also a terminal job. For a terminal job τ_{ij}, the $push$ procedure can determine its new pushed finish time new_ft_{ij} and new pushed start time $new\_st_{ij} = new\_ft_{ij} - e_i$ without any further recursive calls. The **push** operation is formally defined as follows:

push: return the new start time of input jobs after maximum push

```
Input: $\tau_{ij}$
Output: new start time after maximum push $new\_st_{ij}$
begin procedure
1. if $\tau_{ij}$ has no conflicting jobs scheduled up to $d_{ij}$ /* terminal job */
2.     slide $\tau_{ij}$ from $st_{ij}$ to $d_{ij} - e_i$ until MLA violation is observed at $t_{MLA}$ $(t_{MLA} < d_{ij})$.
3.     if $t_{MLA}$ is found, the new finish time $new\_ft_{ij} = t_{MLA}$. otherwise, $new\_ft_{ij} = d_{ij}$.
4.     new start time $new\_st_{ij} = new\_ft_{ij} - e_i$.
5. else /* not a terminal job */
6.     new finish time $new\_ft_{ij} = d_{ij}$.
7.     for each conflicting task $\tau_{i'j'}$ up to $d_{ij}$
8.         $new\_ft_{ij}$ = $min(new\_ft_{ij}$, push($\tau_{i'j'}$)).
9.     end for
```

```
10.     slide $\tau_{ij}$ from $st_{ij}$ to $new\_ft_{ij} - e_i$ until MLA violation
is observed at $t_{MLA}$ ($t_{MLA} < new\_ft_{ij}$).
11.     if $t_{MLA}$ is found, the new finish time $new\_ft_{ij} = t_{MLA}$.
otherwise keep $new\_ft_{ij}$.
12.     new start time $new\_st_{ij} = new\_ft_{ij} - e_i$.
13. end if
14. return $new\_st_{ij}$.
end procedure
```

This algorithm returns the new start time new_st_{ij} after maximally pushing τ_{ij}. If τ_{ij} is a terminal job, its new finished time can be pushed up to its deadline d_{ij} if we could ignore the MLA constraint. In order to consider the MLA constraint, in Line 2, we slide τ_{ij}'s execution interval up to d_{ij} to find a earliest time point t_{MLA} when the MLA constraint can be violated, if any. If such time point t_{MLA} is found, t_{MLA} is the latest possible pushed finish time of τ_{ij} without violating the MLA constraint. Thus, new_ft_{ij} is set to t_{MLA} in Line 3. Otherwise, the new finish time can be pushed up to d_{ij}, that is, $new\_ft_{ij} = d_{ij}$ in Line 3. Once the new finish time is determined, Line 4 can simply calculate the new start time, i.e., $new\_st_{ij} = new\_ft_{ij} - e_i$.

If τ_{ij} is not a terminal job, Lines 7, 8, and 9 recursively call **push** for all dependent jobs to figure out the minimum new start time of all dependent jobs. If we ignore the MLA constraint, the minimum of the deadline d_{ij} and the new pushed start times of all dependent jobs is the latest possible new finish time new_ft_{ij} for τ_{ij}. Lines 10 and 11 can advance the new finish time new_ft_{ij} considering the MLA constraint in the same way as in the terminal job case. With new_ft_{ij}, Line 12 calculates the new start time as $new\_st_{ij} = new\_ft_{ij} - e_i$. Finally, Line 14 returns new_st_{ij}.

Considering new_st_{ij} of all dependent jobs of J_k, we can calculate the maximum slack that can be used for the on-demand request J_k starting from the current scheduling time point t. If the maximum slack is larger than the required execution time e_k and also if executing J_k from t to $t + e_k$ does not violate the MLA constraint, the central regulator sets time t as the start time of J_k and push dependent periodic jobs as needed. The piece of schedule affected by J_k (see "schedule replacement" in Figure 7) is transferred to the corresponding servers so that they can temporarily use the updated schedule piece instead of the original schedule, to accommodate J_k. If the above condition does not hold, the central regulator examines the next scheduling time point to recalculate the maximum slack and so on, until it finds enough slack time during which J_k can be executed without violating the MLA constraint.

C. Distributed Implementation of Scheduling Algorithms

The aforementioned scheduling algorithms assume a central regulator that collects all offline/online measurement requests and builds/updates the global schedule. A centralized regulator is popular in NMIs because it is convenient to initiate and collect measurements into a central database. However, such a centralized mechanism could incapacitate an NMI when there is a failure of the central regulator. Also, some applications require distributed measurement scheduling to gain greater flexibility to dynamically determine the locations of measurement data collection and subsequent analysis. To address these issues, this section presents a mechanism to implement the above scheduling algorithms in a decentralized way.

In a distributed setting, measurement requests (e.g., add/remove periodic measurement tasks and on-demand measurement jobs) arrive at their local servers, possibly concurrently. If each server concurrently updates the schedule upon the arrival of requests,

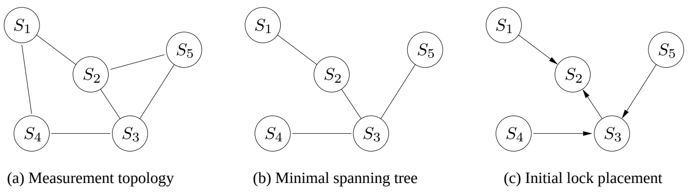


Fig. 8. Minimal spanning tree for the measurement server topology

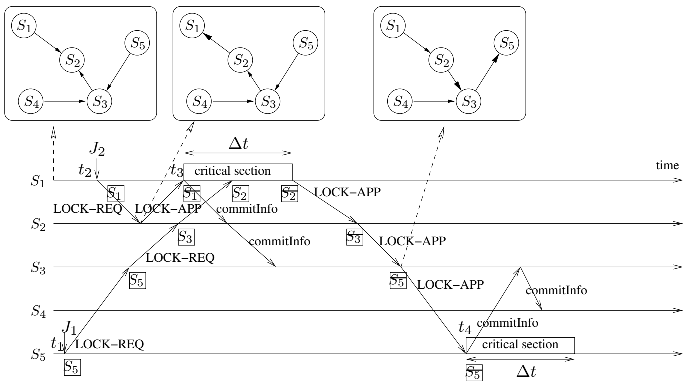


Fig. 9. Distributed schedule update

it breaks the consistency of the schedule and in turn creates measurement conflicts. Therefore, the issue is to serialize the distributed concurrent requests such that the schedule can be updated in a consistent way. For this, we propose to use Raymond's algorithm [16] developed for distributed synchronization. This section describes how Raymond's algorithm works with our scheduling algorithms maintaining the schedule consistency in a distributed way.

For the measurement topology given in Figure 8 (a) as an example, we first create the minimal spanning tree as in Figure 8 (b). This tree is used to maintain a tree-wide single lock with minimal exchange of messages [16]. The basic idea is to allow only the lock holder to commit the arrival of a request at a time, which assures the global serialization of concurrent requests. In the initialization phase, we place the lock at any server, say S_2 in the example of Figure 8 (b), and make each server set its dir variable to the neighbor toward the lock holder as shown in Figure 8 (c).

Upon the arrival of a new request at a server, the server exchanges messages with others along the spanning tree, and eventually gets the lock. Then, it commits the arrival of the request by sending this commitment information to all the affected servers. All the servers that receive this commitment run the same EDF-CE (for an add/remove request of a periodic task) or recursive-push (for an on-demand job) algorithm to update its schedule table. This procedure can be best illustrated by the example of Figure 9. Suppose that the initial lock holder is S_2 as shown in the left-most tree. Also, assume that an on-demand job $J_1(S_5, S_4, Iperf, 10)$ arrives at S_5 at time t_1. Since S_5 is not the lock-holder, it enqueues its ID S_5 in the S_5's queue and sends a LOCK-REQUEST message to the neighbor S_3 pointed by its dir variable. S_3 is not the lock-holder either and thus it enqueues the requester's ID S_5 and sends a LOCK-REQUEST message to the neighbor S_2 pointed by its dir variable. In the meantime, suppose that another request $J_2(S_1, S_3, Iperf, 10)$ arrives at S_1 at time t_2. Since S_1 is not the lock-holder, it enqueues its ID S_1

and sends a LOCK-REQUEST message to S_2 pointed by its dir variable. When S_1's LOCK-REQUEST reaches the lock-holder S_2, S_2's queue is empty and it is not updating the schedule (not in the critical section), and thus it can immediately yield its lock to the requester S_2 by sending a LOCK-APPROVAL message to the requester S_1. It is no longer the lock-holder and set its dir variable to S_1 (see the second tree). When S_1 receives the LOCK-APPROVAL at t_3, it notices that the head of the queue is its own ID and thus can enter the critical section to commit J_2's arrival. Since the J_2's arrival needs to be viewed by all affected servers in a consistent way, S_1 adds the sufficient delay Δt of commitment transmission to t_3 and considers $t_3 + \Delta t$ as the committed arrival time of J_2. Then, S_1 sends the commitment information (J_2 and $t_3{+}\Delta t$) to the all affected servers, S_2 and S_3. Now, S_1, S_2, and S_3 can run the same recursive-push algorithm for inserting J_2 with the same committed arrival time of $t_3 + \Delta t$. When the LOCK-REQUEST message from S_3 arrives at S_2, S_2 is not the lock-holder and its dir is pointing S_1. Thus, the LOCK-REQUEST is forwarded to S_1. When the LOCK-REQUEST reaches S_1, it is the lock-holder but it is already in the critical section to commit J_2. Thus, S_1 enqueues the requester's ID S_2. After that, S_1 leaves the critical section at $t_3 + \Delta t$. At this time, S_1 notices that the head of its local queue is S_2 and thus sends a LOCK-APPROVAL message to S_2 and sets its dir toward S_2. S_2 and S_3 in turn forward the LOCK-APPROVAL and update their dir variables according to the head of their queues until the LOCK-APPROVAL reaches S_5. When S_5 receives the LOCK-APPROVAL at time t_4, it notices that the head of its queue is itself and thus can enter the critical section to commit the arrival of J_1. The commitment phase is the same as that of J_2. As a consequence, the concurrent arrivals of J_1 and J_2 are globally serialized in the order of J_2 and J_1 with the consistent commitment times of $t_3 + \Delta t$ and $t_4 + \Delta t$. Therefore, the schedule can be updated in a globally consistent way, assuring the conflict-free scheduling property.

For the complete and formal description of this distributed schedule update procedure, the readers are referred to [16]. The procedure inherits the proved properties of Raymond's algorithm, such as minimal message exchange for assuring serializability, deadlock-freedom, no-starvation, and fault-tolerance.

D. *Measurement Federation Issues across ISP Borders*

Collecting measurement data within a single ISP domain is not sufficient for distributed computing applications because they often span network paths across multiple ISP domains. For example, application service providers such as Vonage rely on multiple ISPs for delivering word-wide voice over IP (VoIP) and videoconferencing services. To serve their customers meeting the service level agreements (SLAs), ISPs need to support inter-domain measurements that could produce end-to-end Internet measurements. For facilitating such inter-domain measurements, "NMI federations" [17], [5] have emerged where multiple ISPs agree upon a common measurement policy to cooperate with each other.

This section discusses the inter-domain NMI federation issues and explains how our scheduling framework can be incorporated into the federation. For building an NMI federation, all the participating ISPs should agree on the following: (1) sharing each other's measurement server topology, (2) bounding the amount of measurement traffic (i.e., the MLA constraint ψ), (3) authenticated and secure access to measurement resources, and (4) sharing collected measurement data.

First, the measurement server topology of an ISP can be securely revealed only to other ISPs in the same federation using the agreed authentication and encryption methods as will be discussed later. Thus, every measurement server in the NMI federation can have the federation-wide view of the server topology and thus can determine the schedule of measurement tasks even if they span across multiple ISPs. Second, the agreed measurement traffic bound, MLA constraint ψ, can be enforced in our scheduling algorithms as explained in Sections IV-A and IV-B and thus it can be complied across multiple ISPs. Third, for the authenticated and secure access to measurement resources across ISP borders, all ISPs can use a pre-agreed authentication and encryption techniques. For example, upon arrival of a new measurement request, they can use a centralized Kerberos [18] authentication server with Data Encryption Standard or triple DES. This can verify that the requesting domain belongs to the same NMI federation and also prevent intruders from eavesdropping the request for deciphering the authentication mechanism and impersonation as a member of the NMI federation. Finally, the collected measurement data can be shared by multiple ISPs as needed by distributed computing applications, using "Request/Response" schemas being developed by the Global Grid Forum [17].

We envision that the growth of an NMI federation involves mostly political hurdles rather than technical ones. Since the application and ISP communities are realizing the importance of NMI federation for inter-domain distributed computing, we believe that all the political hurdles will be overcome resulting in a world-wide NMI federation. Note that such efforts have been already started by the communities such as Global Grid Forum, Internet2 in USA, and DANTE in Europe [17], [5].

V. Case Study with NWS (Network Weather Service) for Distributed Computing

A. *NWS Network Prediction*

We now apply our measurement scheduling algorithms to the widely-used Network Weather Service (NWS) [9] that can provide network performance forecasts[3]. In this application, one challenge is the gap between the original measurement time requirement of NWS and the actual temporal behavior of our scheduling algorithms. More specifically, NWS periodically issues measurement requests expecting a periodic sampling of network status. However, the scheduler cannot serve the requests exactly at the desired times due to resource conflicts with other measurement requests. As such, any scheduler that tries to avoid conflicts, inevitably creates a jitter in the inter-sampling times of network status. This section presents a simple method for compensating the inter-sampling jitter.

NWS relies on continuous and periodic sampling or *Pure Periodic Sampling* (PPS) of network status. It uses the periodically sampled network status data to maintain the history of network performance, which in turn is used to generate on-going and dynamic network performance forecasts. The forecast time window is the same as the sampling period.

However, it is not always possible for the measurement scheduling algorithm to provide pure-periodic network status data, especially when multiple measurement tasks are running. This can be explained by Figure 10 that shows an example conflict-free schedule of two periodic measurement tasks that have a conflict relation. We can note that the task τ_2 is scheduled pure-periodically with constant inter-sampling times. However,

[3]NWS uses periodically measured network status and forecasts the network performance. Due to its ability of forecasting network performance, NWS has been adopted by a number of networked job schedulers such as AppLes [19], Legion [20], Globus/Nexus [21].

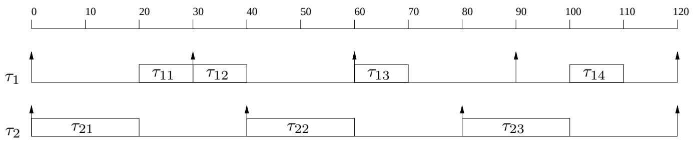


Fig. 10. Jitter of inter-sampling time points of network status

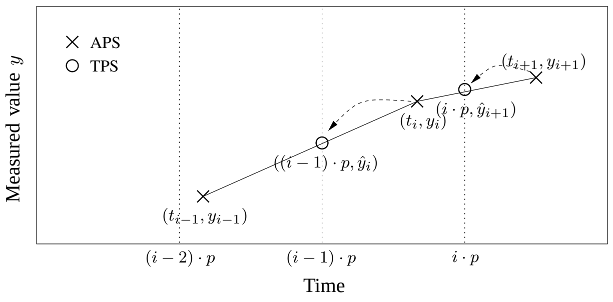


Fig. 11. APS transformation to TPS

the inter-sampling times of task τ_1 vary for every instance. To avoid conflict of multiple concurrent tasks, the actual scheduling time points are inevitably deviated from the periodic release time points by any conflict-free scheduling algorithm. Our EDF-CE also produces such inter-sampling time jitter since it is designed to guarantee the periodic deadlines and not pure periodic execution of jobs. In fact, with our EDF-CE, the inter-sampling jitter of task τ_i can vary from e_i (when a job instance is scheduled just before its deadline and the next one is scheduled at the release time) to $2p_i - e_i$ (when a job instance is scheduled at the release time and the next one is scheduled just before the deadline).

Although our EDF-CE causes inter-sampling jitter between two consecutive jobs, it bounds the jitter by meeting the end-of-period deadlines. Therefore, it can still be used for NWS with simple interpolations of collected network status data. The interpolation is transforming the actual measured data to pure-periodic data using piece-wise linear interpolation. To explain this, let us consider Figure 11. In the figure, (t_{i-1}, y_{i-1}), (t_i, y_i), and (t_{i+1}, y_{i+1}) show the sequence of the $(i-1)$-th, i-th, and $(i+1)$-th *actual periodic sampling* (APS) whose inter-sampling time is not always the same as the period p. To transform the APS sequence to a pure-periodic sampling sequence, which we call *Transformed Periodic Sampling* (TPS), we can draw piece-wise lines between pairs of two APS points . For example, we can draw a line between (t_{i-1}, y_{i-1}) and (t_i, y_i) as shown in Figure 11. With this line, we can estimate the measurement value $\hat{y}_i$ at the pure-periodic i-th sampling time, i.e., $(i-1) \cdot p$. Specifically, $\hat{y}_i$ is given as follows:

$$\hat{y}_i = y_i + \frac{y_{i+1} - y_i}{t_{i+1} - t_i}((i-1) \cdot p - t_i).$$

Thus, the APS data (t_i, y_i) with inter-sampling jitter can be transformed to the pure-periodic TPS data $((i-1) \cdot p, \hat{y}_i)$ as shown in Figure 11. Similarly, (t_{i+1}, y_{i+1}) can be transformed to $(i \cdot p, \hat{y}_{i+1})$. Now, the NWS can use the TPS data rather than the original measured data to provide the network performance forecast. With this simple interpolation method, we will show in Section VI-C that our EDF-CE can work well with NWS to produce accurate forecasts.

B. Use of NWS for Distributed Computing

Due it its ability to forecast the network status, NWS can be used for a number of distributed computing applications that rely on networked computational resources. In this section, we sketch the scenarios where the NWS based on our conflict-free measurements can help two typical examples of distributed computing, i.e., Grid computing and videoconferencing.

In Grid computing, users often transfer computational jobs involving large data sets (sometimes even on the scale of tera bytes) to remote computing sites [22]. If proper network paths are not selected, then jobs that traverse problematic paths could hold up the speedy completion of other queued jobs that traverse problem-free paths and thus significantly degrade the overall efficiency of Grid computing. This problem can be avoided by using the NWS. It can accurately monitor and predict the network status by using the conflict-free (and so no-misleading) measurement data. The job scheduler of Grid computing can use such network status data to select the computing sites and network paths in a way that ensures the optimal efficiency of the overall computing. In addition, the ability of network status prediction can allow the job scheduler to dynamically change the network path selections before severe performance degradation happens.

In videoconferencing, interactive sessions involving three or more participants are established using call-admission controllers that manage Multi-point Control Units (MCUs). MCUs combine the admitted voice and video streams from participants and generate a single conference stream that is multicast to all the participants. If a call admission controller selects problematic network paths between the participants and MCUs, the perceptual quality of the conference stream could be seriously affected by impairments such as video-frame freezing, audio drop-outs, and even call-disconnects. Using the NWS, such a problem can be avoided. The call admission controllers can consult the NWS to figure out the network paths that can satisfy the application QoS requirements. In addition, the network status forecasts from NWS can also be used to monitor whether the current selection will experience problems due to the paths that may soon degrade the application QoS severely. In such cases, the call admission controllers can dynamically change to alternate network paths that are identified to satisfy QoS requirements for the next forecasting period.

The selected paths in both cases of Grid computing and videoconferencing can be enforced in Internet by using MPLS explicit routing or by exploiting path diversity based on multi-homing or overlay networks [23], [24].

VI. Experimental Results

In this section, we evaluate the performance of our measurement scheduling algorithms. We first perform simulations with synthetic measurement tasks to show the maximum schedulability by the EDF-CE algorithm and the average response times of on-demand requests by the recursive pushing algorithm. Then, we present performance evaluation results on an actual Internet2 testbed. Finally, we present the case study results of applying our EDF-CE to NWS.

A. Performance Evaluation Results using Synthetic Tasks

Our synthetic task set is comprised of four periodic active measurement tasks τ_1, τ_2, τ_3 and τ_4. The period p_i of each task τ_i is randomly generated from [1000 sec, 10000 sec]. The execution time e_i of each task τ_i is randomly generated from [100 sec,

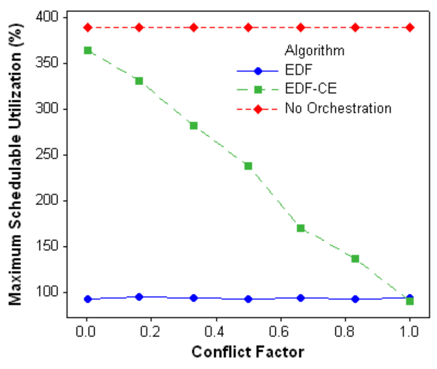


Fig. 12. Maximum schedulable utilization by three scheduling algorithms

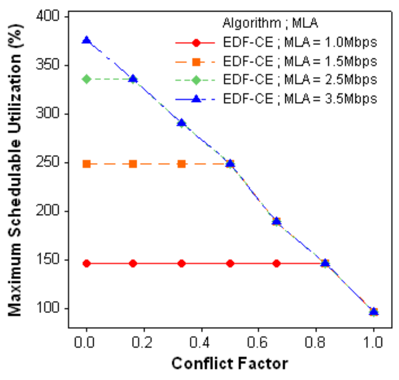


Fig. 13. Effect of MLA ψ and conflict factor to EDF-CE

999 sec]. Since the measurement topology and inter-task conflict relations can be represented by a task conflict graph, we conduct this experiment as changing only the task conflict graph. The task conflict graph of the four tasks is randomly created using a parameter called a *conflict factor*. The conflict factor represents the probability that there is a conflict edge between any two tasks. Therefore, when the conflict factor is 1, the task conflict graph is fully connected. If the conflict factor is 0, there is no edge between tasks.

For each sample task set and task conflict graph, we use the "maximum schedulable utilization" $\sum_{i=1}^{4} e_i/p_i$ as the performance metric. We determine the maximum schedulable utilization by gradually increasing execution times e_i until the scheduling algorithms fail to construct a feasible schedule.

We compare three scheduling algorithms:

- **No-orchestration** that schedules measurement jobs at their release times without considering measurement conflicts,
- **EDF** that schedules only one measurement job at a time using the non-preemptive EDF algorithm just like a single processor EDF scheduling, and
- **EDF-CE** that is proposed in this paper.

Figure 12 shows the maximum schedulable utilization as increasing the conflict factor. Here, we assume a large MLA ψ, say 50 Mbps, and thus avoid any MLA bottlenecks when finding the schedule. Each plotted point in the figure is the average of 1000 random sample task sets. EDF's maximum schedulable utilization is constantly bounded under 100% regardless of the conflict factor since it does not allow concurrent execution even when possible. On the other hand, our EDF-CE algorithm can maximally utilize the concurrent execution whenever possible. When the conflict factor is zero, EDF-CE allows concurrent execution of all four tasks. This is similar to scheduling the four tasks on four independent processors. Thus, the maximum schedulable utilization reaches up to 400%. As the conflict factor increases, the maximum schedulable utilization gradually decreases. When the conflict factor is 1, i.e., when all four tasks conflict with each other, EDF-CE automatically degenerates to the single processor EDF and hence gives the maximum schedulable utilization of 100%. The result shows that EDF-CE is leveraging the "maximal but only possible" concurrent execution by explicitly considering the conflict dependency among tasks. The no-orchestration approach always gives the maximum schedulable utilization of 400% since all four tasks can be concurrently executed ignoring the conflict dependency. This, however, causes many conflicts as will be shown in Section VI-B resulting in many misleading reports of actual network performance.

Figure 13 illustrates how the maximum schedulable utilization of EDF-CE is bounded by the MLA constraint ψ and conflict factor. As expected, a higher value of ψ accommodates a larger number of concurrent jobs and hence produces a higher maximum schedulable utilization. For a given ψ value, the maximum schedulable utilization is constant up to a certain point of the conflict factor and then starts decreasing. Such a trend explains that ψ is the bottleneck when the conflict factor is small, whereas the conflict dependency becomes the bottleneck when the conflict factor is large.

To study the performance of the "recursive push" algorithm for handling on-demand measurement requests, we simulate random arrivals of on-demand jobs and schedule them over the offline EDF-CE schedule. The offline specified task set consists of four periodic tasks as before, and their execution times and periods are randomly generated from [1 minute, 10 minutes] and [20 minutes, 200 minutes], respectively. The execution times and inter arrival times of on-demand jobs are also randomly generated from [1 minute, 10 minutes] and [20 minutes, 200 minutes], respectively. The performance metric is the average of the response times for 1000 on-demand jobs. We compare our recursive push algorithm with a background approach that schedules an on-demand job in the earliest gap present in the offline EDF-CE schedule within which the on-demand job can execute to completion. Figure 14 shows that our recursive push algorithm can significantly improve the responsiveness for on-demand measurement requests. Note that the average response time in both the background and recursive push cases increases as the conflict factor increases. This is because a higher conflict dependency among tasks reduces the concurrent execution of jobs and thus reduces the gaps available to schedule the on-demand jobs.

To estimate the overhead of online scheduling, we measure the algorithm running time for each on-demand job on 2.4 GHz Pentium 4 Linux PC. Figure 15 shows the average times as increasing the number of periodic tasks, while fixing the conflict factor as 0.8. Even for a large number of periodic tasks with a high conflict factor, our recursive push algorithm can find the slack and calculate the updated schedule within tens of milliseconds. This is a negligible delay comparing with typical measurement task execution times in the order of minutes.

In order to study the overhead of the distributed implementation of the scheduling algorithms, we simulate both centralized and distributed implementations of the recursive push algorithm in a large scale network. For the network topology, we use the Waxman topology with 1000 nodes produced by the BRITE tool [25]. From the topology of 1000 nodes, we randomly select N nodes as the measurement servers creating a "measurement topology" with N measurement servers over the network topology

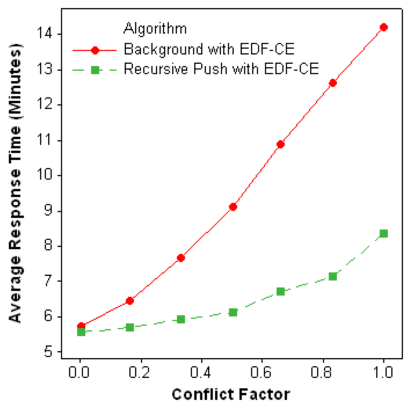


Fig. 14. Average response time of on-demand jobs

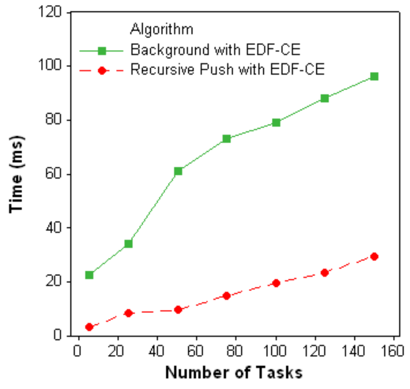


Fig. 15. Online schedule overhead for on-demand jobs

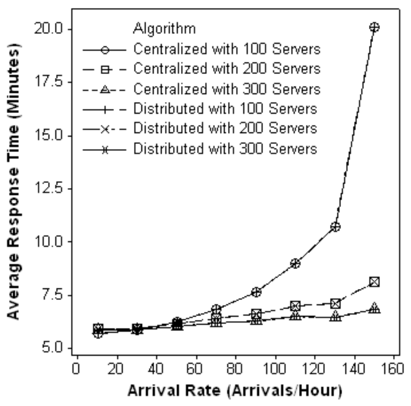


Fig. 16. Response time comparison of the centralized and distributed implementations

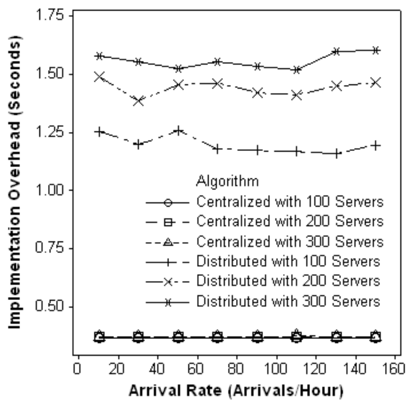


Fig. 17. Implementation overhead comparison of the centralized and distributed implementations

with 1000 nodes. We consider three different Ns: 100, 200, and 300. These choices represent NMIs with a reasonably large number of servers noting that the largest NMI deployment today, i.e., the NLANR AMP project [3] has around 150 measurement servers deployed all over the world. On top of the measurement topology, we use a synthetic task set with 100 offline periodic measurement tasks. The period p_i and the execution time e_i of each task τ_i are randomly generated from [20 minutes, 200 minutes] and [1 minute, 10 minutes], respectively. Then, the execution times of all 100 tasks are scaled such that the total utilization $\sum_{i=1}^{100} e_i/p_i$ become 50%. Each task is assigned with randomly selected src and dst servers. With this offline periodic task set, we generate the offline schedule using the EDF-CE algorithm. Given the offline schedule, we simulate the random arrival of 1000 on-demand jobs with random execution times following the exponential distribution with the average 5 minutes. We conduct the simulation as we increase the average arrival rate from 10 jobs/hour to 150 jobs/hour following the Poisson distribution. Each on-demand job is assigned with randomly selected src and dst servers. In the following figures, we report the average of 100 simulation runs.

Figure 16 compares the average response times of on-demand jobs by the centralized and distributed implementations of the recursive-push algorithm. The centralized and distributed implementations show almost the same response times. This is because the total message passing delay to transfer the lock in the distributed implementation is at most 2 seconds even in a large scale measurement topology with 300 servers as shown in Figure 17. Also, such delay does not increase with the increase of the arrival rate. This is due to the message minimization capability of Raymond's algorithm as the number of requests increases [16]. Another interesting observation in Figure 16 is that the response time is smaller when the number of servers is larger. This is because the on-demand job workload is scattered over a larger number of servers and hence the per-server workload is smaller.

B. Performance Evaluation Results on an Internet2 Testbed

We have actually implemented and deployed our scheduling algorithms in an NMI that is being used to monitor network paths on the Internet2 Abilene network backbone. The scheduling framework consists of a "Scripting Language Interface" and a central regulator as shown in Figure 18. The scripting language interface provides a generic and automated way to input measurement specifications such as measurement server topology, periodic measurement tasks, and MLAs. These specifications are interpreted by the central regulator to construct schedule timetables for the measurement servers. The constructed schedule timetables are transferred to the corresponding servers to initiate the measurement jobs at the planned times.

Our Internet2 testbed has five sites each of which is equipped with a measurement server as shown in Figure 19(a). To collect the actual measurement data, we run five periodic measurement tasks as shown Figure 19(b). The resulting task conflict graph is shown in Figure 19(c).

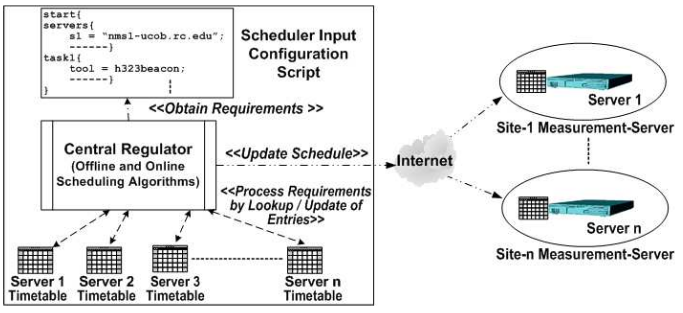

```
start{
servers{
    s1 = "nms1-ucob.rc.edu";
    ------}
task1{
    tool = h323beacon;
    ------}
}
```
Scheduler Input Configuration Script
<<Obtain Requirements >>
Central Regulator (Offline and Online Scheduling Algorithms)
<<Update Schedule>>
<<Process Requirements by Lookup / Update of Entries>>
Internet
Server 1
Site-1 Measurement-Server
Server n
Site-n Measurement-Server
Server 1 Timetable
Server 2 Timetable
Server 3 Timetable
Server n Timetable

Fig. 18. Structure of Measurement Scheduling Framework

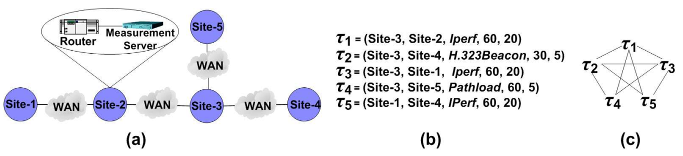

τ_1 = (Site-3, Site-2, *Iperf*, 60, 20)
τ_2 = (Site-3, Site-4, *H.323Beacon*, 30, 5)
τ_3 = (Site-3, Site-1, *Iperf*, 60, 20)
τ_4 = (Site-3, Site-5, *Pathload*, 60, 5)
τ_5 = (Site-1, Site-4, *IPerf*, 60, 20)

Fig. 19. Internet2 Testbed Setup

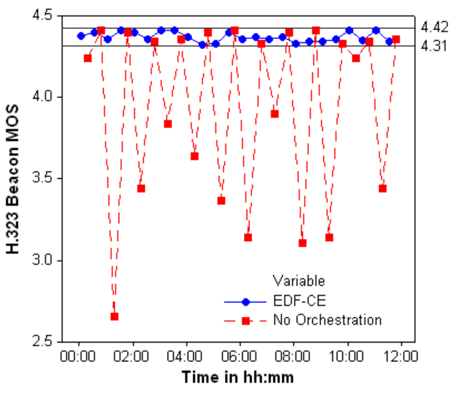


Fig. 20. H.323 Beacon MOS measurements between Site-3 and Site-4

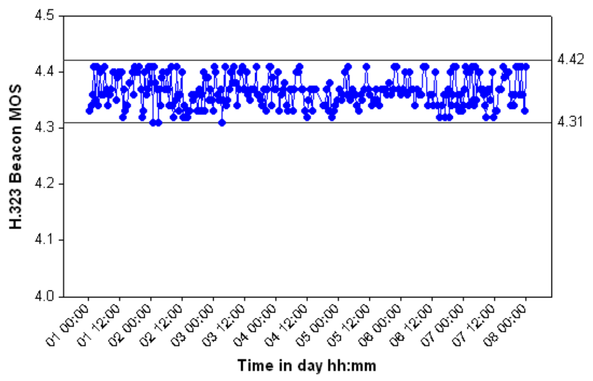


Fig. 21. H.323 Beacon MOS measurements over a week period between Site-3 and Site-4

Figure 20 shows the H.323 Beacon MOS reports[4] measured between Site-3 and Site-4 by task τ_2. To compare EDF-CE and No-Orchestration, we pick the same 12-hour time frames in two consecutive days. For the 12-hour time frame of the first day, we use No-Orchestration method to run all five measurement tasks in Figure 19(b) and collected the MOS reports from τ_2. For the 12-hour time frame of the second day, we use EDF-CE and collect the same reports. From these two experiments, we can observe that the proposed EDF-CE guarantees zero conflict while No-Orchestration causes 50% instances of τ_2 to overlap with other tasks. All the overlaps in the No-Orchestration schedule are indeed conflicts since all the tools used in Figure 19(b) are CPU intensive and channel intensive. In terms of MOS accuracy, however, we are not sure which curve is better reflecting the reality of the network status, since we do not know the "true-real" network status. In order to have a good representation of the reality of the network status between Site-3 and Site-4, we run only τ_2 over a week long period. The results are shown in Figure 21. From the figure, we can affirm that MOS fluctuation between 4.31 and 4.42 is natural in reality between Site-3 and Site-4. The MOS values in Figure 20 collected using EDF-CE well match the representation of the reality in Figure 21. In contrast, the MOS reports by the no-orchestration method in Figure 20 show much larger fluctuation, which seems abnormal comparing with Figure 21. We can conclude that these abnormal fluctuations are due to 50% instances of τ_2 conflicting with other tasks.

Although we do not present the data for Iperf of τ_1, τ_3, τ_5 and Pathload of τ_4 due to the page limit, we observed the similar measurement anomalies in No-Orchestration but not in EDF-CE. From these observations, we can justify the importance of measurement orchestration for the correct estimation of network status.

C. Case Study Results with NWS

In this section, we show how well our EDF-CE can work in combination with NWS. For this purpose, we use actual trace data obtained from NWS measurements for a path traversing a T1 connection with a total bandwidth 1.5 Mbps [26]. The trace data corresponds to "hourly" samples of available bandwidth on the T1 line over a two-day period. We assume that the trace data reflects the "actual" network performance trend on the path. Using this trace data, we generate two sample sequences, one representing the ideal pure-periodic sampling (*PPS*) with a period of 2 hours, and another one that corresponds to actual sampling (*APS*) by EDF-CE. The 2-hour-based PPS is obtained by considering every other sample in the trace data assuming that no other monitoring tasks are present. This 2-hour-based PPS is the ideal measurement sequence expected by NWS for 2-hour look-ahead forecasts. The

[4]MOS measurements reported by the H.323 Beacon are based on the E-Model [2], which is a computational model standardized by ITU-T to estimate the perceptual user quality for VoIP. The MOS values are reported on a quality scale of 1 to 5; [1, 3) range being poor, [3, 4) range being acceptable and [4, 5] range being good. MOS values close to 4.41 are desirable for high-quality VoIP.

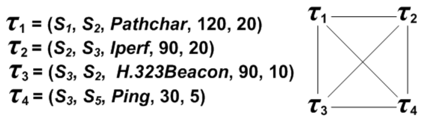

τ_1 = (S_1, S_2, *Pathchar*, 120, 20)
τ_2 = (S_2, S_3, *Iperf*, 90, 20)
τ_3 = (S_3, S_2, *H.323Beacon*, 90, 10)
τ_4 = (S_3, S_5, *Ping*, 30, 5)

Fig. 22. Four task example for determining inter-sampling times of APS data

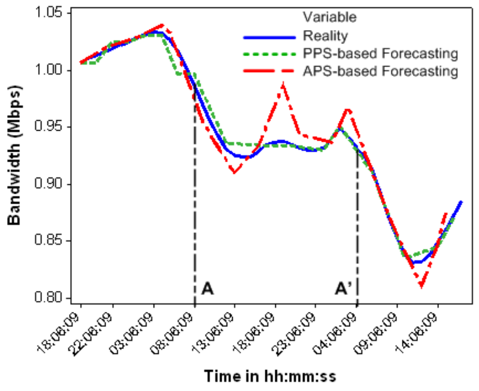


Fig. 23. Comparison of Forecasts of PPS and APS

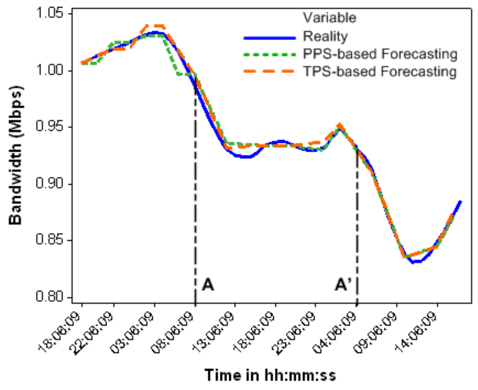


Fig. 24. Comparison of Forecasts of PPS and TPS

actual measurement sequence, APS, however, inevitably deviates from the PPS due to scheduling of conflicting tasks. To model the APS, we simulate the EDF-CE with four measurement tasks in Figure 22. Note that task τ_1 serves NWS by providing 2-hour based sampling of available bandwidth. This simulation provides the inter-sampling time distribution of τ_1, which is used to select the samples from the trace data. The selected samples approximately represent the actual sequence of samples obtained by τ_1 as scheduled by EDF-CE in the presence of three other tasks[5].

Figure 23 shows the reality of the available bandwidth (i.e., one hour based trace data), the NWS forecasted bandwidth using PPS, and the same using APS. The NWS forecasting using the ideal PPS closely match the actual trend. However, the NWS forecasting using APS has non-negligible differences from the reality. This is because of inter-sampling time jitter caused by EDF-CE. This problem can be fixed by a simple transformation of sampled data as described in Section V. Figure 24 shows that the NWS forecasting using the transformed samples denoted by TPS can be very close to the ideal forecasting by PPS.

VII. Conclusion and Future Work

In this paper, we identify the measurement conflict problem, which results in misleading measurements of network status when multiple conflicting measurement tools are executing at the same time on the same server or path. From the observation, we formulate the measurement scheduling problem as a real-time scheduling problem.

For the optimal schedulability of periodic measurement tasks, we use the EDF principle, which has been proven to be optimal in single processor preemptive scheduling and perform well in general settings. Our significant enhancement is to leverage concurrent execution, which clearly differentiates the measurement scheduling problem from the classical real-time scheduling problems. Our enhanced EDF algorithm called EDF-CE allows concurrent execution of multiple measurement jobs not only on the isolated servers and paths but also on the same server and path, as long as they do not conflict—no misleading reports. This significantly improves the schedulability and thus allows us to run measurements more frequently or saves significant time for on-demand requests.

We also propose an online scheduling algorithm to serve on-demand measurement requests as early as possible. The online algorithm can steal the maximum slack without violating any periodic deadlines and thus can almost immediately schedule the on-demand requests. Therefore, the response times of on-demand requests can be significantly reduced compared to their background processing.

Our proposed scheduling algorithms have actually been implemented and deployed on the Internet2 Abilene network. The actual experimental results demonstrate the pertinence and trustworthiness of our proposed scheduling algorithms.

[5]Due to the one hour based granularity of the original trace data, the sequence of selected samples is only an approximation with quantization errors up to one hour. However, it is acceptable in terms of showing how the aforementioned simple interpolation can resolve the inter-sampling time jitter caused by EDF-CE, which ranges from 0 to 4 hours.

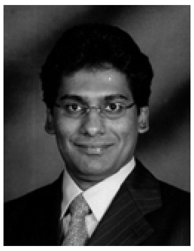
Acknowledgment

This work has been supported in part by Research Settlement Fund for the new faculty of SNU, The Ohio Board of Regents and the American Distance Education Consortium.

References

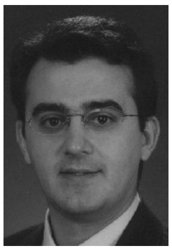
[1] P. Calyam, C.-G.Lee, P. K. Arava, and D. Krymskiy. Enhanced EDF Scheduling Algorithms for Orchestrating Network-wide Active Measurements. In *Proc. of IEEE RTSS*, 2005.
[2] P. Calyam, W. Mandrawa, M. Sridharan, A. Khan, and P. Schopis. H.323 Beacon: An H.323 Application related End-to-end Performance Troubleshooting Tool. In *Proc. of ACM SIGCOMM NetTs*, 2004.
[3] T. McGregor, H.-W. Braun, and J. Brown. The NLANR Network Analysis Infrastructure. In *IEEE Communications Magazine*, 2000.
[4] P. Calyam, D. Krymskiy, M. Sridharan, and P. Schopis. TBI: End-to-end Network Performance Measurement Testbed for Empirical Bottleneck Detection. In *Proc. of IEEE TRIDENTCOM*, 2005.
[5] E. Boyd, J. Boote, S. Shalunov, and M. Zekauskas. The Internet2 E2E piPES Project: An Interoperable Federation of Measurement Domains for Performance Debugging. Technical report, Internet2 Technical Report, 2004.
[6] A. Tirumala, L. Cottrell, and T. Dunigan. Measuring end-to-end Bandwidth with Iperf using Web100. In *Proc. of Passive and Active Measurement Workshop*, 2003.
[7] A. Downey. Using Pathchar to estimate Internet link characteristics. In *Proc. of ACM SIGCOMM*, 1999.
[8] C. Dovrolis, P. Ramanathan, and D. Moore. Packet Dispersion Techniques and Capacity Estimation. *IEEE/ACM Transactions on Networking Journal*, 2004.
[9] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 1999.
[10] J. Liu. *Real-Time Systems*. Publication of Prentice Hall, 2000.
[11] S. Kalidindi and M. Zekauskas. Surveyor: An Infrastructure for Internet Performance Measurements. In *Proc. of INET*, 1999.
[12] M. Alves, L. Corsello, D. Karrenberg, C. Ogut, M. Santcroos, R. Sojka, H. Uijterwaak, and R. Wilhelm. New Measurements with the RIPE NCC Test Traffic Measurements Setup. In *Proc. of Passive and Active Measurements Workshop*, 2002.

[13] B. Gaidioz, R. Wolski, and B. Tourancheau. Synchronizing Network Probes to avoid Measurement Intrusiveness with the Network Weather Service. In *Proc. of IEEE High-performance Distributed Computing Conference*, 2000.
[14] H. Wang, D. Zhang, and K. Shin. Change-Point Monitoring for Detection of DoS Attacks. *IEEE Transactions on Dependable and Secure Computing*, 2004.
[15] P. Calyam, C.-G. Lee, P. K. Arava, D. Krymskiy, and D. Lee. OnTimeMeasure: A Scalable Framework for Scheduling Active Measurements. In *Proc. of IEEE E2EMON*, 2005.
[16] K. Raymond. A Tree-based Algorithm for Distributed Mutual Exclusion. *ACM Transactions on Computer Systems*, 1989.
[17] GGF NMWG Request/Response Schema, 2006. URL:`nmwg.internet2.edu`.
[18] J. Steiner, C. Neuman, and J. Schiller. Kerberos: An authentication service for open network systems. In *Proc. of USENIX*, 1998.
[19] F. Berman and R. Wolski. Scheduling from the Perspective of the Application. In *Proc. of High-Performance Distributed Computing Conference*, 1996.
[20] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Reynolds. Legion: The Next Logical Step Towards a Nationwide Virtual Computer. Technical report, University of Virginia Technical Report CS-94-21, 1994.
[21] T. Defanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY: Wide Area Visual Supercomputing. *International Journal of Supercomputer Applications*, 1996.
[22] D. Reed and C. Mendes. Intelligent Monitoring for Adaptation in Grid Applications. In *Proc. of the IEEE*, 2005.
[23] R. Prasad, M. Jain, and C. Dovrolis. Effects of Interrupt Coalescence on Network Measurements. In *Proc. of Passive and Active Measurement Workshop*, 2004.
[24] J. Han and F. Jahanian. Impact of Path Diversity on Multi-homed and Overlay Networks. In *Proc. of IEEE DSN*, 2004.
[25] Boston University. BRITE: Representative Internet Topology Generator, 2006. http://www.cs.bu.edu/brite.
[26] Middleware Initiative. NWS User's Guide, 2006. http://archive.nsf-middleware.org/documentation/NMI-R5/0/gridscenter/NWS/users_guide.htm.



Prasad Calyam received the BS degree in Electrical and Electronics Engineering from Bangalore University, India, and the MS degree in Electrical and Computer Engineering from The Ohio State University, in 1999 and 2002, respectively. Currently, he is a Ph.D. Candidate in Electrical and Computer Engineering at The Ohio State University. He is also currently a Senior Systems Developer/Engineer at OARnet, a division of the Ohio Supercomputer Center. His current research interests include network management, active/passive network measurements, voice and video over IP and network security. He is a student member of the IEEE.

Chang-Gun Lee received the BS, MS and Ph.D. degrees in Computer Engineering from Seoul National University, Korea, in 1991, 1993 and 1998, respectively. He is currently an Assistant Professor in the School of Computer Science and Engineering, Seoul National University, Korea. Previously, he was an Assistant Professor in the Department of Electrical and Computer Engineering, The Ohio State University, Columbus from 2002 to 2006, a Research Scientist in the Department of Computer Science, University of Illinois at Urbana-Champaign from 2000 to 2002, and a Research Engineer in the Advanced Telecomm. Research Lab., LG Information and Communications, Ltd. from 1998 to 2000. His current research interests include real-time systems, complex embedded systems, ubiquitous systems, QoS management, wireless ad-hoc networks, and flash memory systems. Dr. Lee is a member of the IEEE Computer Society.



Eylem Ekici received his BS and MS degrees in Computer Engineering from Bogazici University, Istanbul, Turkey, in 1997 and 1998, respectively. He received his Ph.D. degree in Electrical and Computer Engineering from Georgia Institute of Technology, Atlanta, GA, in 2002. Currently, he is an Assistant Professor in the Department of Electrical and Computer Engineering of The Ohio State University, Columbus, OH. Dr. Ekici's current research interests include wireless sensor networks, vehicular communication systems, and next generation wireless systems, with a focus on routing and medium access control protocols, resource management, and analysis of network architectures and protocols. He is an associate editor of Computer Networks Journal (Elsevier) and ACM Mobile Computing and Communications Review. He has also served as the TPC co-chair of IFIP/TC6 Networking 2007 Conference.

Mark Haffner received the BS degree in Electrical Engineering from University of Cincinnati in 2006. Currently, he is pursuing an MS degree in Electrical and Computer Engineering at The Ohio State University. His current research interests include active/passive network measurements, RF circuit design and software-defined radios. He is a student member of the IEEE.

Nathan Howes is pursuing a BS degree in Computer Science and Engineering at The Ohio State University. His current research interests include active/passive network measurements and network security. He is a student member of the IEEE.