Task-Execution Scheduling Schemes for Network Measurement and Monitoring

Zhen Qin, Roberto Rojas-Cessa, and Nirwan Ansari

Department of Electrical and Computer Engineering
New Jersey Institute of Technology
University Heights, Newark NJ 07102.

Abstract

Measurement is a required process in high performance networks for efficient quality-of-service (QoS) provisioning and service verification. Active measurement is an attractive approach because the measurement traffic injected into the network can be controlled and the measurement tasks can be distributed throughout the network. However, the execution of measurement tasks in common parts of a network may face contention for resources, such as computational power, memory, and link bandwidth. This contention could jeopardize measurement accuracy and affect network services. This contention for limited resources defines a conflict between measurement tasks. Furthermore, we consider two sets of measurement tasks, those used to monitor network state periodically, called periodic tasks, and those for casual measurements issued as needed, called on-demand measurement tasks. In this paper, we propose a novel scheduling scheme to resolve contention for resources of both periodic and on-demand measurement tasks from graph coloring perspective, called ascending-order of the sum of clique number and degree of tasks. The scheme selects tasks according to the ascending order of the sum of clique number and conflict task degree in a conflict graph and allows concurrent execution of multiple measurement tasks for high resource utilization. The scheme decreases the average waiting time of all tasks in periodic measurement tasks scheduling. For on-demand measurement tasks, the proposed scheme minimizes the waiting time of inserted on-demand tasks while keeping time space utilization high. In other words, the total time spent on finishing all the tasks is shortened. We evaluate our proposed schemes under different measurement task assignment scenarios through computer simulations, and compare the performance of this scheme with others that also allow concurrent task execution. The simulation results show that the proposed scheme produces effective contention resolution and low execution delays.

Key words: Scheduling, Network Measurement, Active Measurement, Graph Coloring, Clique, Quality of Service, QoS, List Coloring

1. Introduction

Some applications, such as voice over IP (VoIP), streaming video and online gaming have stringent requirements for Quality-of-Service (QoS) provisioning, which further requires accurate and up-to-date information of the network performance, through measuring and monitoring tools to estimate and collect those data. Therefore, network measurement becomes an important subject driven by Internet Service Providers (ISPs) to quantify network status, monitor existing traffic, and service verification on service agreement compliance for applications with QoS requirements.

Measurement techniques can be coarsely divided into passive and active approaches. Passive measurement uses traversing traffic, whether carrying users' data or network control packets, to determine the network state. The accuracy of passive measurement is a function of the amount of existing traffic. On the other hand, active measurement has controllable properties that are independent of

Email address: {zq4, rojas, ansari}@njit.edu
(Zhen Qin, Roberto Rojas-Cessa, and Nirwan Ansari)

the absence of user traffic, thus making it an attractive approach [1]-[12]. In active measurement, a measurement point, which can be a router or end host or some equipment attached to them, creates and sends probing packets to the target (destination) measurement point with controlled departure times. Either the destination measures the arrival time in a synchronized network, or the source estimates the delay time by using the response of the destination point [13]-[16]. Figure 1 shows an example of a network with active measurement for (a) end-to-end (using end hosts) paths, or (b) local links (between neighbor routers). Without loss of generality, Figure 2 shows a measurement infrastructure designed by Internet2 E2E piPEs projects [18]. The network information obtained by active measurements can be, for example, available bandwidth, capacity, one-way delay, round-trip time (RTT), jitter, and topology data. The adoption of active measurement can be found in several large-scale networks [20]-[23].

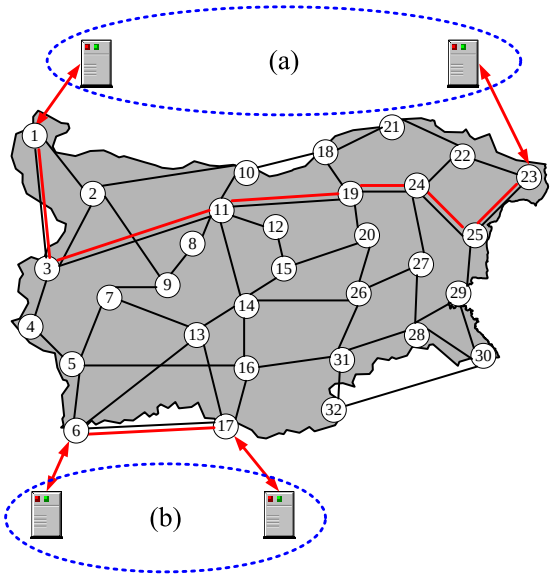

Figure 1: Network measurement implementation topology.

Examples of active measurement tools that can be deployed in any network in general, range from the simple ones, such as Ping and Traceroute, to the more sophisticated, such as Pipechar [24], Pathload [25], Cing [26], Clink [27], Nettimer [28], Pathrate [29], Pathchar [30], and Yaz [31], among others. A network measurement toolkit includes the various measurement tools to evaluate the different QoS parameters. The toolkit shown in Table 1, though

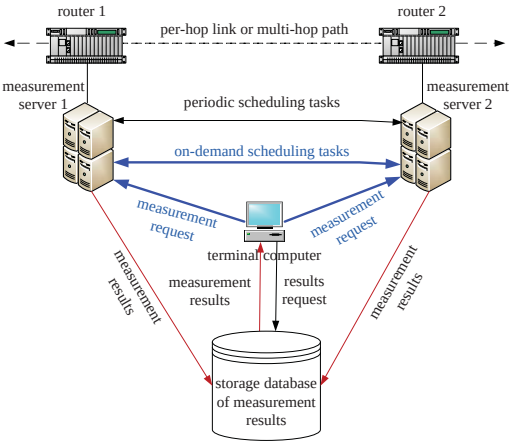

Figure 2: An example of network measurement infrastructure.

Table 1: Selected measurement tools for QoS parameters.

QoS Parameter	Tool
One-way delay	OWAMP[32]
Round-trip delay	Ping
Available bandwidth	Pipechar, Pathload
Topology	Traceroute
Bandwidth capacity	Pathchar

not an exclusive one, is an example. Different tools require different network resources and execution times. The measurement tools are normally run periodically to monitor the consecutive network status, by which the tasks are called periodic measurement tasks. These tasks are known in advance and can be scheduled before their execution starts. Moreover, the tools can also be invoked once (or for short periods) at any arbitrary time to measure a network parameter as needed. These tasks are referred to as on-demand tasks. On-demand tasks emerge at any time and need to be scheduled in combination with the periodic tasks while the network is in operation.

Independently of the measurement approach used, probing overhead is a general concern for active measurement mechanisms as it may affect the user traffic. For example, an active measurement experiment [19] showed that a 700-Byte packet size used in 60-packet probing trains can achieve sufficiently accurate results of available bandwidth measurement per path on the Internet. In this case, one path overhead is about 42KB, and so measuring all end-to-end paths in a 200-nodes bidirectional mesh system requires about 1.7GB for just

Table 2: Comparison of network consumption of sample tools.

Tool	CPU/ Memory	Bandwidth	Time
Ping	very low	very low	≤2s
Pipechar	Low	Low	≥20s
Pathload	Low	Medium	7s

one snapshot if all network links are simultaneously tested. Therefore, the network resources need to be efficiently managed under active probing.

In addition, distributed measurement tasks may be executed simultaneously at one measurement point in a network. Hence, it is possible that different measurement tasks contend for network resources, including transmission channels and bandwidth. Measurement processes that are executed in different common points also contend for resources, such as processing time, bandwidth, and memory. The accuracy of some measurement processes may be affected by other measurement processes run concurrently. This contention for resources is called measurement conflict problem. To gain insight of the implications of contention for resources, we executed Pipechar, Pathload, and Ping in a host, all at the same time, and used them to measure several parameters in the transmission from one host to another through a 100-Mbps fast Ethernet link [16]. We observed that the measurement resources and measurement processing time had a large discrepancy among those measurement tools, as shown in Table 2, and the obtained measurement results are instable because of the disturbance from other measurement processing. Sommers and Barford [17] also implemented a testbed through which the experiment results show that the measurements of packet loss and delay from active probes can be skewed significantly due to the contention of probing packets. Thus, the active measurement tasks that are performed at each measurement point need to be scheduled to avoid both potential resource contention and measurement disturbance from each other while achieving a satisfactory measurement in terms of time and accuracy.

To solve the above problem, we provide a solution to schedule the periodic and on-demand measurement tasks to achieve the following four goals:

1. Avoid conflicts among concurrent executed measurement tasks.
2. Network resources are not exhausted by measurement tasks.
3. Shorten the waiting time of each measurement task for the execution.
4. Shorten the total completion time of measurement tasks set, that is, improve the resource utilization.

To comply with the above requirements, we propose an algorithm to schedule periodic tasks and to improve the measurement efficiency. We also propose an algorithm to schedule on-demand measurement tasks that minimize the delay of both periodic tasks and the incoming on-demand tasks. Both algorithms are based on graph coloring theory, where each measurement task is treated as a vertex in a graph, and the contention/conflict by two tasks is represented as an edge connecting those two vertices.

The remainder of this paper is organized as follows. Section 2 analyzes the contention problem of distributed network measurement tasks. Section 3 introduces related scheduling algorithms for network measurement. Section 4 describes the modeling of measurement tasks for scheduling by using a coloring approach. Section 5 introduces our proposed scheduling schemes for periodic and on-demand measurement tasks, respectively. Section 6 shows the performance evaluation obtained by computer evaluation, and analysis of the proposed schemes and other comparable schemes. Section 7 presents our conclusions.

2. Problem Analysis

According to the classification approach of scheduling introduced by Graham *et al.* [33], the task scheduling problem is defined in terms of a three-tuple classifications $[\alpha, \beta, \gamma]$, where α defines the machine (processor) environment, β specifies the job's characteristics, and γ denotes the optimality criterion. Following this classification method, the measurement scheduling problem can be described as $[P, \{rec, r_i\}, \sum C_i]$. Here, P is the number of identical parallel processors to perform the required jobs. However, different from that approach [33], P is a variable instead. The value of P depends on the number of measurement tasks run simultaneously. Considering that n tasks need to be processed, the following relationship exists:

$$P \leq n \tag{1}$$

rec refers to the constraints on the resources used by the execution of measurement tasks. In order to minimize or to avoid the impact of probing packets on the performance of regular data traffic, a network resource constraint, such as the maximum bandwidth, is set at each measurement point. This is called measurement resource constraint (MRC) in this paper. Scheduling measurement tasks need to ensure that the total amount of resources consumed by the measurement tasks are within this constraint *rec*. Measurement task i is denoted as τ_i in the remainder of this paper. The parameter r_i denotes the release time of a measurement task τ_i, upon which one instance of the task τ_i becomes available for processing or execution. $\sum C_i$ indicates that the optimal criterion chosen is to minimize the total completion time on P parallel processors, where C_i denotes the completion time of the measurement task τ_i. This optimal criterion reflects the fourth goal listed in Section 1. It is easy to see that the third goal listed in Section 1 is the sufficient and necessary condition of the fourth goal, as described by Lemma 1. Therefore, $\sum C_i$ can cover both the third and fourth goals.

Lemma 1. *Minimizing the total completion time of a set of measurement tasks is equivalent to minimizing the average waiting time of the measurement tasks in this set.*

Proof. For a measurement tasks set, the completion time of task τ_i is:

$$C_i = e_i + w_i \tag{2}$$

where e_i is the execution time of measurement task τ_i and w_i is the waiting time of task τ_i. Hence, the total completion time of the measurement tasks set is:

$$\begin{aligned} \sum C_i &= \sum e_i + \sum w_i \\ &= \sum e_i + m \times w_{avg} \end{aligned} \tag{3}$$

where m is the number of measurement tasks in the set and w_{avg} is the average waiting time of the tasks. Since the execution time of each measurement task is a constant, the sum of the execution time $\sum e_i$ is a constant too. According to Equation 3, minimizing $\sum C_i$ is equal to minimizing $m \times w_{avg}$, and thus is equal to minimizing w_{avg}. □

A scheduling algorithm can be further classified as preemptive or non-preemptive. In preemptive scheduling, the execution of a task can be interrupted prior to completion and resumed later. On the other hand, in non-preemptive scheduling, a task must be executed to completion once execution has started. In general, measurement task scheduling is regarded as non-preemptive scheduling as the measurement results are expected at completion and the measurement results may be time sensitive. Another issue with this problem that differentiates it from the others is the potential conflict that measurement tasks have with each other. This characteristic increases the complexity of the scheduling scheme because the tasks cannot be just sorted according to one parameter (e.g., deadline or execution time of the task), but also the conflict with scheduled tasks has to be considered.

3. Related Work

Round robin is one of the simplest scheduling schemes [22, 34, 35] where the tasks are executed by a fixed order in uni-processor systems and only one task is executed at a time. This scheme requires the longest processing time for measurement tasks as it does not admit concurrent execution.

Network Weather Service (NWS), a well-known network measurement infrastructure, adapts a token passing scheme [36] to ensure mutual exclusion between measurement tasks. In this scheme, the measurement point that receives a token is entitled to execute a measurement task. Afterwards, the measurement point releases the token to a successor. However, this method does not allow concurrent execution of measurements.

Deadline driven scheduling (DSS), also known in the literature as the Earliest Deadline First (EDF) scheduling scheme [37], selects tasks based on their deadlines, and was originally defined for uni-processor execution.

It is shown that the problem of determining whether a given periodic task system is non-preemptively feasible on either a single processor or multiprocessors is NP-hard in a strong sense [38], [39]. To provide network measurement scheduling, a scheduling algorithm based on EDF that allows multiple concurrent executions, referred to as EDF-CE [40], was recently proposed. This approach initializes a queue that stacks all pending tasks to be processed in an EDF order, where the deadline is defined as the time before the task must be executed again. Whenever a task is ready to be released or a task finishes execution, the available tasks in

the queue are scheduled. This method introduces the possibility of overlapping multiple tasks in some time slots, but it does not consider the utilization ratio; in other words, sorting the tasks in the pending queue with their deadlines ignores the fact that the concurrent execution of multiple tasks greatly depends on the existing conflicts between the tasks as much as on the tasks' deadlines.

4. Modeling of Network Measurement Scheduling Schemes

4.1. Definitions

Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ represent the measurement tasks set with up to n measurement tasks to be executed in the network. Here, τ_i is characterized by a three-tuple of parameters:

- $a(\tau_i)$: the time the measurement task is released, which is the task's arrival time.
- $e(\tau_i)$: the execution time required by a measurement task to complete the measurement.
- $p(\tau_i)$: the period of the measurement task, or the time to execute task τ_i after the previous instance. This parameter describes how often a measurement task is executed.

A timetable of periodic measurements is constructed by sequences of tasks, each of which is executed again in $p(\tau_i)$ units of time, and each task requires execution of $e(\tau_i)$ time units. The j_{th} job (or repetition) of measurement task τ_i is denoted as τ_{ij}. Thus, the first job, τ_{i1}, of measurement task τ_i occurs at time $a(\tau_i)$; consecutive jobs generated by τ_i occur exactly $p(\tau_i)$ time units apart. Figure 3 illustrates an example delineating the terms defined above.

In a set of periodic tasks where the tasks (and the number of them) do not change and where each task can have any particular period, the combination of tasks' release times is finite. This is, after a long period of time, because of the task periodicity, the combination of release times repeats again. Therefore, for the measurement set τ, we define the term hyperperiod p_h to be the period of time where all tasks in the set occur at different times and without replication of the combination of release times. That is, all periodic tasks in one hyperperiod are able to follow the same schedule as used in the previous hyperperiod. The hyperperiod is defined as

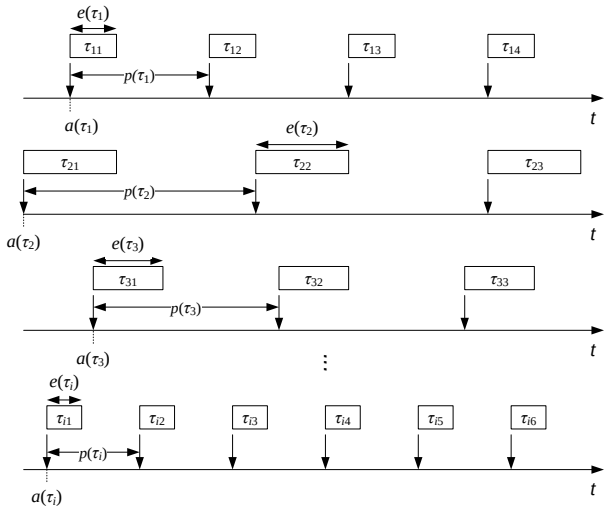

Figure 3: Illustration of network measurement tasks.

the least common multiple of the periods of all measurement tasks in τ.

$$p_h = lcm\left(p(\tau_1), p(\tau_2), \ldots, p(\tau_n)\right) \tag{4}$$

Without loss of generality, we define the execution time $e(\tau_i)$, initial available time $a(\tau_i)$, and the period $p(\tau_i)$ as integer multiples of a time unit which is referred to as a time slot. The deadline of each job $d(\tau_{ij})$ coincides with the period, that is, the job τ_{ij} should be completed before the next job $\tau_{i(j+1)}$ is available to be executed. According to this definition, Lemma 2 can be readily obtained:

Lemma 2. *Given a measurement tasks set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, at any time instance, there is at most one job available to be executed for any measurement task $\tau_i \in \tau$.*

Proof. At any time instance, there must be a job available for execution at the beginning of that period. If there are some jobs generated from previous periods still pending for execution, those postponed jobs passed their own deadlines and they are considered as missed jobs. Hence, there is at most one job for each measurement task at any time-instance. □

4.2. Modeling of Measurement Scheduling

Our scheduling algorithms are based on graph theory. In the literature, there are some articles using graph coloring to solve time slots assignment problem [41, 42, 43], but most of them are designed

for single processing, which are not fit for multi-task processing such as the network measurement scenario.

Consider a measurement tasks set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ to be executed in a network. Each measurement task can be represented as a node ($\in V$) in a graph and any two measurement tasks are connected by a link ($\in E$) if they are to be executed with mutual exclusion on the measurement point or channel. These tasks are said to be adjacent to each other. The graph $G(V, E)$ that describes these nodes and links is called a conflict graph. Figure 4 illustrates an example of a conflict graph where two measurement tasks are to be executed between measurement points 1 and 2 in a full-duplex connection. Assume that task τ_1 contends with τ_2 for the available memory at measurement point 1, and at the same time, it contends for the transmission channel with τ_3. Task τ_3 also contends with τ_4 for available memory at measurement point 2. Therefore, these four tasks comprise a conflict graph with three links. In this example, measurement tasks τ_1 and τ_4 (represented by shaded nodes), or τ_2 and τ_3 (represented by unshaded nodes) can be concurrently executed.

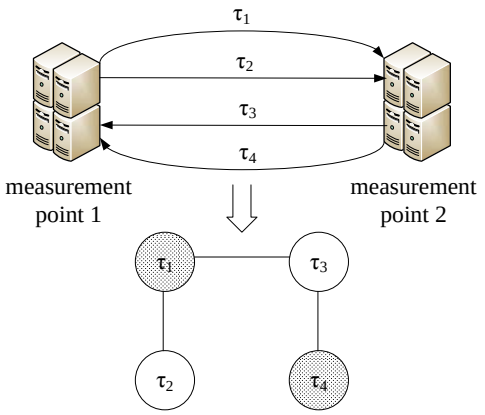

Figure 4: Illustration of the relationship between measurement tasks by a conflict graph.

In our considered network, there is a central controller to compute the schedule of all measurement tasks and to send out the schedule information to each measurement point. This central management mode is feasible and adopted in real network measurement frameworks. Scheduling is requested each time when a new job is available for execution and when a job execution has been completed. We name these time instances as scheduling points. There is a waiting queue to store the jobs available for execution. At each scheduling point, jobs stored in the waiting queue become eligible candidates for the scheduler. Based on the conflict relationship between the measurement tasks, these jobs that belong to different measurement tasks construct a conflict graph at the job level. The conflict relationship between jobs follows the same conflict relationship between measurement tasks. For periodic tasks, the conflict relationship among them is known prior to performing scheduling because the submitted tasks and the amount of resources they consume are both known in advance. For on-demand tasks, the attributes of measurement tools are *a priori* so the tasks' conflicts are known once an on-demand task emerges.

As the measurement results obtained by earlier periodic measurement tasks are used to describe the current network performance, it is desired that the measurement tasks can be completed as soon as possible after a task is available for execution. Therefore, the scheduling problem is converted into a process to schedule the available jobs at each scheduling point so as to minimize the job waiting time for execution. At the same time, the scheduling of measurement jobs at one scheduling point is enunciated as the arrangement of the vertices of graph G at the job level such that none of the nodes connected with each other are scheduled for simultaneous execution. This process can be described as a *vertex coloring problem* as follows.

Scheduling of Measurement Tasks: *Given a conflict graph $G(V, E)$ with vertices $V = V(G)$, assign each vertex a color out of the range $[1, 2, \ldots, k]$ such that no two adjacent vertices have the same color.*

Here, each color maps to one time slot. The color set to be used by a vertex v_{ij} in the conflict graph is mapped to the time range $[t_c, d(\tau_{ij})]$ as described by Equation 5, where t_c is the current scheduling point and $d(\tau_{ij})$ is the deadline of the job mapped by vertex v_{ij}. That is, the scheduler only considers the time slots prior to a job's deadline.

$$[1, 2, \ldots, k] \rightarrow [t_c, d(\tau_{ij})] \tag{5}$$

Each measurement point is considered to have limited processing and storage (memory) capabilities, and each channel to have a limited bandwidth capacity. Therefore, the load of intrusive probing packets in active measurement needs to be restricted within a range, so as to minimize the dis-

turbance of the measurement of the existing data traffic, as described by MRC values. We propose to use a consumption matrix to describe such constraints. Let us denote the number of schedule slots and the number of the measurement jobs as the column and row of a matrix as shown in Figure 5. The resource utilization objective can be described as follows:

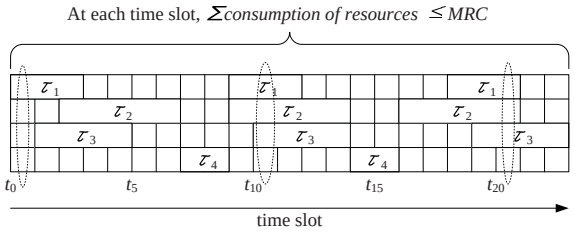

Figure 5: Consumption matrix.

Resource Utilization of Measurement Tasks*: Jobs of measurement tasks set* $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ *with execution times* $e(\tau_1)$, $e(\tau_2)$, $\ldots$, $e(\tau_n)$, *can be represented as a* $p_h \times n$ *consumption matrix A, where a row indicates the task and its duration in time slots and the column indicates the time slot. The maximum number of rows is bounded by the amount of processing resources constrained by Equation 1. Each column as circled in Figure 5 represents the consumption of network resources at that particular time slot.*

The objective is to place the measurement tasks in the consumption matrix such that $\sum_{j=1}^{n} A_{ij} \leq MRC, \forall i \in [1, 2, \ldots, p_h]$, where p_h is the hyper-period duration, i.e., the total consumption of resources by measurement tasks per time slot is within the measurement resource constraint.

5. Proposed Scheduling Schemes

This section introduces our scheduling schemes for periodic and on-demand measurement tasks. The following definitions are used in the description of the proposed schemes.

- Clique: a maximal set of adjacent vertices of graph G.
- Clique number: the number of vertices in the largest clique of G, denoted as $\omega(G)$.
- Degree: degree of vertex v in graph G is the number of adjacent vertices of v in G, denoted as $d_G(v)$; the maximum degree of graph G is the largest number of $d_G(v)$, and it is denoted as $\Delta(G)$.

5.1. Periodic Measurement Tasks Scheduling Scheme

Following the model of the scheduling problem described in Section 4.2, our proposed algorithms consider the jobs stored in the waiting queue for scheduling at each scheduling point. If a job can be scheduled in the time range $[current\ scheduling\ point, deadline\ of\ job]$ without any conflict with already scheduled jobs at any given time slot, this job is removed from the waiting queue and the corresponding time slots for execution are marked in the consumption matrix; otherwise, the job is kept in the waiting queue and waits for consideration at the next scheduling point. Hence, the goal is to find a feasible scheme to schedule the maximum number of concurrent jobs at each scheduling point, so that the most time space in the consumption matrix can be utilized.

Consider available jobs in the waiting queue. Since their execution times are integer multiples of a time slot and the time slot can be mapped to a vertex, each task can be divided into a set of sub-vertices as follows:

In a conflict graph $G(V, E)$ *at the job level, each vertex* v_{ij} *that maps job* τ_{ij} *has a set of sub-vertices* $(\tau_{ij}^1, \tau_{ij}^2, \ldots, \tau_{ij}^\alpha)$, *where* α *is the length of* $e(\tau_{ij})$ *in time slots.*

As the sub-vertices of v_{ij} represent the different but consecutive time slots of a task, they are said to contend with each other (or to have a conflict with each other). These conflicts can be described by a complete sub-graph G_{ij} as in the example shown in Figure 6. Conflict graph G is further represented by its sub-vertices and it is denoted as $G^s(V^s, E^s)$. The clique number of a sub-graph G_{ij} is equal to the number of vertices in G_{ij}. Here, G^s is the graph constructed by sub-vertices.

As each color represents one time slot, each sub-vertex in graph G^s is a candidate for a color assignment, so that any two adjacent sub-vertices must not possess the same color. Each sub-vertex is restricted to allowed colors that satisfy the relationship denoted by Equation 5. This is called the list coloring problem. To solve this problem, we propose to sort the sub-vertices in the ascending order

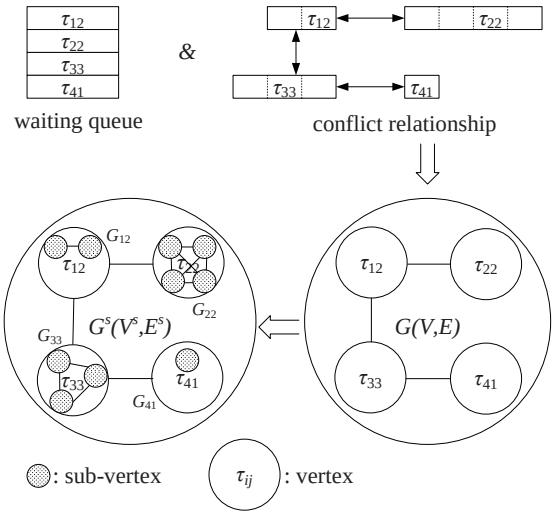

Figure 6: Example of sub-graph.

of their degree in graph G^s:

$$d^s_G(v^l_{ij}) = d_G(v) + \omega(G_{ij}) = d_G(v) + e(\tau_{ij}), \quad \forall v^l_{ij} \in G_{ij} \quad (6)$$

The rationale to schedule jobs in this fashion is the expectation that a sub-vertex with a small degree has a few conflicts; therefore, a large number of tasks might be scheduled at the same time. In a network with a measurement scheduling environment, this can be described by two aspects. For a sub-vertex v^l_{ij} and its adjacent sub-vertex v_x:

- v^l_{ij} and v_x map to the same job: Then, the low degree implies the job has a short execution time. This part is represented as the execution time of the vertex $e(\tau_{ij})$, or by the clique number of the sub-graph $\omega(G_{ij})$. Scheduling a job with a short execution time will leave more available time slots for other jobs in the waiting queue.

- v^l_{ij} and v_x map to different job: Then, the low degree of the sub-vertex indicates the job might have few conflicts with other available jobs in the waiting queue. Scheduling a job with few conflicts allows additional jobs to be executed concurrently, thus increasing the resource utilization.

The scheduling procedure is described below:

Step 1. At current scheduling point t_c, check if there is a new job available for execution. If so, the new job is placed in the waiting queue.

Step 2. Map the candidate jobs in the waiting queue to a conflict graph G and convert G into sub-graph G^s.

Step 3. Sort the sub-vertices in the ascending order of their degree, as described by Equation 6.

Step 4. Schedule the first job as indicated by the sorted sequence. Any sub-vertex v^l_{ij} selected to be scheduled will be colored with other sub-vertices belonging to the same job τ_{ij} with consecutive colors. The used colors are the intersection set as $\overline{colors_{in-conflict}} \cap [t_c, d(\tau_{ij})]$ where $[t_c, d(\tau_{ij})]$ is the time interval from t_c to $d(\tau_{ij})$, $colors_{in-conflict}$ is the set of available colors possessed by the on-going conflict jobs, and $\overline{colors_{in-conflict}}$ is the complementary set of $colors_{in-conflict}$, i.e., the available colors that can be used by v^l_{ij}.

Step 5. Check if the colored job and other on-going jobs violate the resource constraint MRC. If there is no violation, remove the colored job from the waiting queue, remove the corresponding sub-vertices from the sorted sequence, and add the completion time of the job to the scheduling point list.

Step 6. Color the next sub-vertex in the sorted sequence. Repeat Steps 4 to 5.

Step 7. Go to the next scheduling point. Repeat Steps 1 to 6.

The algorithm of periodic measurement-tasks scheduling is described by the pseudo code in Figure 7.

5.2. On-Demand Measurement Tasks Scheduling Scheme

During the execution of the periodic measurement, a network administrator may request sporadic on-demand measurement tasks to test specific network performance parameters at a particular time. Furthermore, on-demand tasks might conflict with some periodic or on-demand tasks. Each on-demand task has also defined execution and deadline times, and it is considered with either a priority higher than or equal to that of the scheduled periodic tasks. The proposed scheduling scheme for on-demand measurement tasks is able to handle both of these two cases adaptively. The goal of scheduling on-demand tasks with higher priority is to execute the on-demand tasks as soon as

Periodic Tasks Scheduling Algorithm

Input: measurement task set τ
  3-tuple parameters of tasks $a(\tau_i)$, $e(\tau_i)$, $p(\tau_i)$
  tasks conflict matrix F
  measurement resource constraint MRC
Output: start time of jobs T
Initialize hyperperiod p_h
Initialize scheduling point list S
Initialize waiting queue $Q = \{\}$
while S is not empty
  if new available job $\tau_{new} \in Q$
    remove the old instance of τ_{new} due to expiration
  end
  $Q = Q + \tau_{new}$
  set up conflict graph with sub-vertices $f : (F, Q) \rightarrow G^s(V^s, E^s)$
  sort Q by ascending order of degree of sub-vertices $d_{G^S}(v)$
  for each job τ_{ij} in sorted Q
    find the available colors range R
    if consecutive sequence $L = e(\tau_{ij})$ exists in R
      & consumption of τ_{ij} and on-going jobs satisfies MCR
      assign the first long enough consecutive colors L to τ_{ij}
      $Q = Q - \tau_{ij}$, record start time of τ_{ij} in T
      update scheduling point list:
        $S = S + completion\ time\ of\ \tau_{ij}$
    end
  end
Go to next scheduling point
end

Figure 7: Pseudo code of scheduling algorithm for periodic measurement tasks.

possible while minimizing the latency of the periodic tasks caused by the insertion of on-demand tasks. On the other hand, scheduling on-demand measurement tasks with the same priority as periodic tasks aims to shorten the average waiting time for all measurement tasks including on-demand and periodic tasks.

The proposed method schedules all the tasks with higher priority first, and then schedules the remaining on-demand and periodic tasks according to the ascending order of the degree of sub-vertices, as explained below:

Step 1. When a new on-demand task arrives at t_c, check the priority type of the on-demand task. If its priority is high, store this on-demand task to the waiting queue of high priority tasks Q_{high}. If the priority is equal to that of the periodic tasks, the on-demand task is stored to $Q_{regular}$.

Step 2. Schedule all the candidate jobs in the waiting queue of high priority tasks Q_{high}. In the pre-computed schedule, all the jobs of periodic tasks that finish their execution before t_c and the jobs that are still being executed at time t_c are discarded/cancelled. The jobs that start processing after t_c are considered as rescheduled. Follow Steps 2 to 6 of the previous scheduling procedure for periodic tasks. Note that the scheduling points are updated so the completion time of the scheduled jobs in Q_{high} are added into the scheduling points list. After this step, all the possible jobs in Q_{high} must be either scheduled or expired because there are no available time slots to be scheduled before the job's deadline.

Step 3. Add those jobs that start processing after t_c in the pre-computed schedule to the waiting queue of regular priority tasks $Q_{regular}$. Schedule all candidate jobs in $Q_{regular}$ following the previous scheduling procedure for periodic tasks.

Figure 8 shows an example to illustrate this scheduling procedure. In this example, the on-demand task τ_{od} conflicts with periodic tasks τ_1 and τ_3, as shown in Figure 8.a. If the priority of τ_{od} is higher than that of other periodic tasks, then when it arrives at t_c, all periodic jobs that start the execution after t_c are stored in $Q_{regular}$ while τ_{od} is stored in Q_{high}. Thus, τ_{od} is the first to obtain a schedule. As shown in Figure 8.b, τ_{od} is first scheduled and only the schedule of job τ_{32} is changed. If τ_{od} has same priority as other periodic task, then τ_{od} and all periodic jobs that start the execution after t_c are stored in $Q_{regular}$ and sorted in the ascending order of sub-vertices' degree. As shown in Figure 8.c, τ_{od} is scheduled with longer waiting time than in Figure 8.b, but rescheduling for other periodic jobs is unnecessary.

The algorithm of on-demand measurement tasks scheduling is described by the pseudo code shown in Figure 9.

5.3. Computational Complexity Analysis

According to Lemma 2, there are at most n jobs in the waiting queue if there are n tasks in the measurement tasks set. Using a simple sorting algorithm such as binary tree sort, the computational

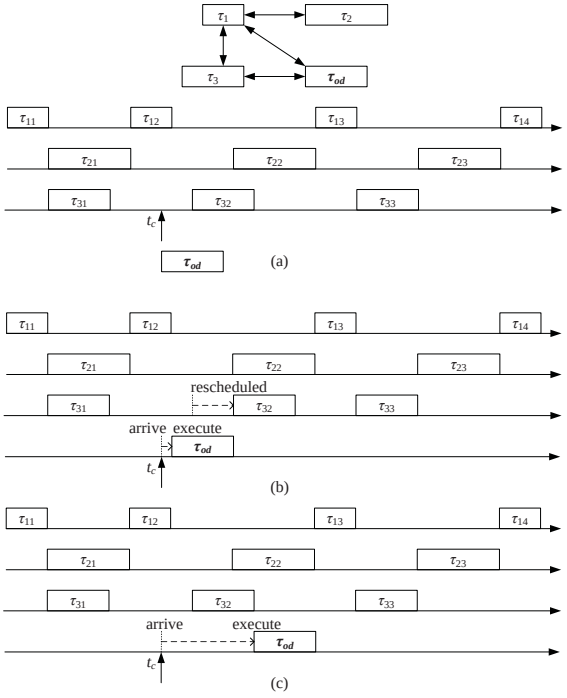

Figure 8: Example of scheduling on-demand measurement task: (a) pre-computed schedule; (b) on-demand task has higher priority; (c) on-demand task has same priority as periodic tasks.

complexity of sorting n jobs is $n\lg(n)$. In one hyperperiod, assume there are m scheduling points which indicate the time jobs arrive, then the computational complexity of the proposed algorithm is $mn\lg(n)$. Let's denote C as the number of unique completion times of all jobs and K as the total number of jobs to be executed in a hyperperiod. Therefore, we get:

$$K = \sum_{i=1}^{n} \frac{p_h}{p(\tau_i)}$$

and so the following relationship exists:

$$m \leq \sum_{i=1}^{n} \frac{p_h}{p(\tau_i)} + C \leq 2\sum_{i=1}^{n} \frac{p_h}{p(\tau_i)}$$

Therefore, the computational complexity of the proposed algorithm is $n\lg(n)\sum_{i=1}^{n}\frac{p_h}{p(\tau_i)}$, and thus the complexity can be decreased by limiting the upper-bound of p_h. Some previously proposed methods aimed to achieve this goal [44], but this is out of scope of this paper.

On-demand Tasks Scheduling Algorithm

Input: measurement task set τ
  3-tuple parameters of tasks $a(\tau_i)$, $e(\tau_i)$, $p(\tau_i)$
  tasks conflict matrix F
  on-demand task τ_{od}
  measurement resource constraint MRC
  pre-computed schedule T_o
Output: start time of jobs T
Initialize hyperperiod p_h
Initialize scheduling point list S
Initialize waiting queue $Q_{high} = \{\}$, $Q_{regular} = \{\}$
if priority of τ_{od} is high
  $Q_{high} = Q_{high} + \tau_{od}$
elseif priority of τ_{od} is regular
  $Q_{regular} = Q_{regular} + \tau_{od}$
end
set up conflict graph with sub-vertices:
  $f : (F, Q_{high}) \rightarrow G^s_{high}(V^s_{high}, E^s_{high})$
sort Q by ascending order of degree of sub-vertices $d_{G^S_{high}}(v)$
schedule each job in Q_{high} and update scheduling point list
set up conflict graph with sub-vertices:
  $f : (F, Q_{regular}) \rightarrow G^s_{regular}(V^s_{regular}, E^s_{regular})$
sort Q by ascending order of degree of sub-vertices $d_{G^S_{regular}}(v)$
schedule each job in $Q_{regular}$ and update scheduling point list

Figure 9: Pseudo code of scheduling algorithm for on-demand measurement tasks.

6. Simulation Results

To study the performance of the proposed algorithms, we compared them with other scheduling algorithms.

6.1. Schemes for Comparison

We considered algorithms able to process multiple measurement tasks at the same time for a fare comparison to the proposed algorithms for their execution on an infrastructure with sufficient resources. All of these algorithms have the same computational complexity as the proposed ones. These algorithms are described next:

6.1.1. Round-Robin

We improve the original round robin scheme to empower it with the concurrent execution capability. The improved scheme selects tasks for execution by following a pre-defined order. The scheme performs scheduling at each scheduling point. At a scheduling point, all the available jobs waiting to be scheduled are selected in a pre-defined round-robin

order. If there is no conflict with current on-going task, the job is scheduled; otherwise, the job is kept in the queue to be considered/scheduled at the next scheduling point. This algorithm is described in Figure 10.

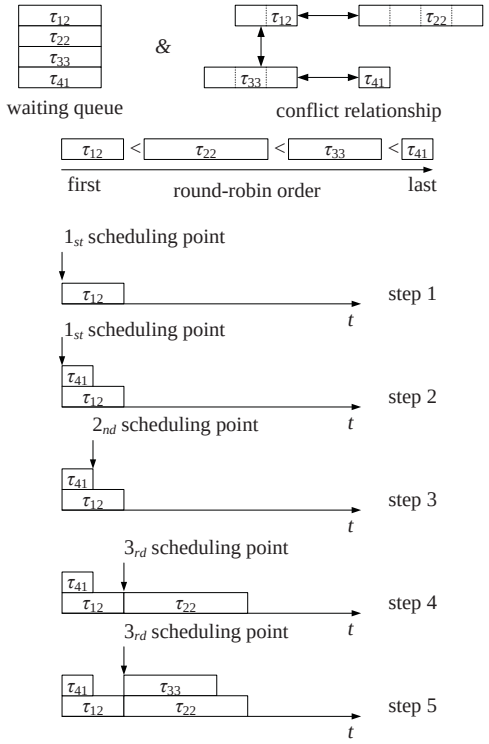

Figure 10: Illustration of the improved round robin scheduling algorithm.

6.1.2. Descending Order of Sub-Vertices' Degree (DOSD)

This scheme, also introduced here for comparison purposes, follows a similar procedure as described in Section 5.1 for the ascending order version, except that this scheme sorts the jobs in the waiting queue in the descending order of the degree of the sub-vertices mapped to the jobs, in Step 3.

6.2. Evaluation Method

The algorithms are compared in terms of the average normalized waiting time of all jobs in one hyperperiod that is defined as below:

$$Avg.\ normalized\ waiting\ time = avg(\sum \frac{w(\tau_{ij})}{p(\tau_{ij})})$$

where $w(\tau_{ij})$ is the waiting time of the job τ_{ij}. $w(\tau_{ij})$ is formally defined as the difference between the time that the job starts execution and the beginning time the job is available to be executed. In the worst case, some measurement jobs may be missed due to time expiration (i.e., the waiting time exceeds the task period). We define the waiting time of the missed job equal to its period time.

As the network performance is monitored by periodic measurement requests, the measurement jobs are expected to be scheduled at desired sampling times that the interval time between any two consecutive samplings is a constant. However, because of the conflict of the network measurement tasks, the measurement jobs are scheduled at the time deviated from the desired sampling times. The average normalized waiting time is used to reflect how severe such deviation impacts the acceptance of the measurement sampling results. For example, if a measurement task with period equal to 20 minutes waits for 1.5 minute to start execution, the measurement result is still acceptable to be used as periodic samples. However, if a measurement task with period 2 minutes waits for 1.5 minute for execution, the measurement sample obtained is far from the expected measurement sampling time.

Another evaluation parameter is the execution success ratio of jobs to be executed, which is defined as:

$$Execution\ success\ ratio = \frac{number\ of\ executed\ jobs\ in\ one\ hyperperiod}{number\ of\ total\ jobs\ in\ one\ hyperperiod}$$

6.3. Simulation Results of Periodic Tasks Scheduling

In this simulation, the period of the periodic measurement tasks is uniformly distributed in the range of [11,100] time units, and the execution times of the periodic measurement tasks are uniformly distributed in the range of [2,10] time units. The initial time of task $a(\tau_i)$ is randomly selected in the range of [1,5] time units. We observed the performance of algorithms for different conflict probability values from 0 to 1.0 with increments of 0.05. A conflict probability of 0 between two tasks means that there is no conflict between them, therefore there is no edge connecting these two vertices in the conflict graph. A conflict probability of 1.0 means that there is a conflict between any two tasks, corresponding to a fully connected conflict graph. There might be a high conflict probability in a real network where the ongoing measurement tasks demand

network resources for exclusive use. As an example, the simultaneous measurement of bandwidth, delay, jitter and other parameters at a gateway in a small network could be a network performance bottleneck as all measurement tools contend for the memory, processing time, and uplink/downlink bandwidth of that gateway. To observe the maximum performance of the scheduling schemes, the measurement resource is assumed to be large enough so there is no MRC constraint on measurement tasks. We compare the performance of the algorithms with 10 and 20 periodic tasks scenario. The simulation is run 1000 times (i.e., for each time a random tasks set and the conflict relationship are generated) for each scenario.

Figure 11 shows the average normalized waiting times of 10 periodic tasks for these schemes. The figure shows that the proposed scheme has the lowest average normalized waiting times, and EDF-CE has the highest.

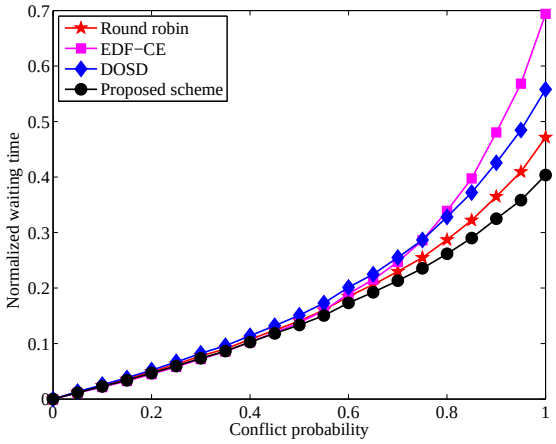

Figure 11: Normalized waiting time for 10 periodic measurement tasks.

Figure 12 shows the success ratio of 10 periodic tasks of the compared schemes. The figure shows that as the conflict probability increases, the success ratio of the schemes decreases. Here, the success ratio of the proposed scheme is the highest among other schemes as this scheme misses scheduling the fewest number of tasks as compared to the other schemes, while DOSD, which sorts the task in the opposite order, has the lowest success ratio. The combination that increases the success ratio seems to be the selection of a small task and with a small number of conflicts.

Figure 13 shows the normalized waiting times of these schemes with 20 tasks. The outcome for 20 tasks is similar to the case with 10 tasks, where the proposed scheme achieves the lowest waiting time. The advantage of using the proposed scheme is more pronounced for scenarios with a larger number of tasks.

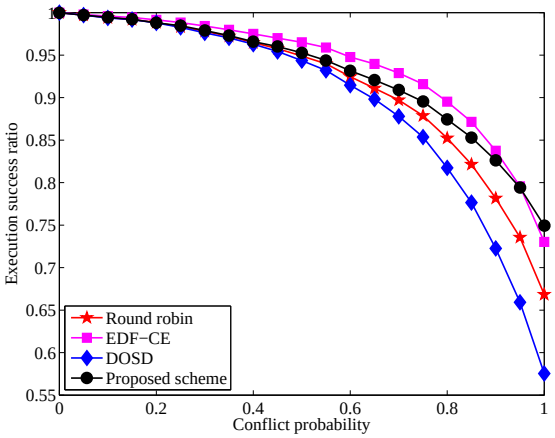

Figure 12: Execution success ratio for 10 periodic measurement tasks.

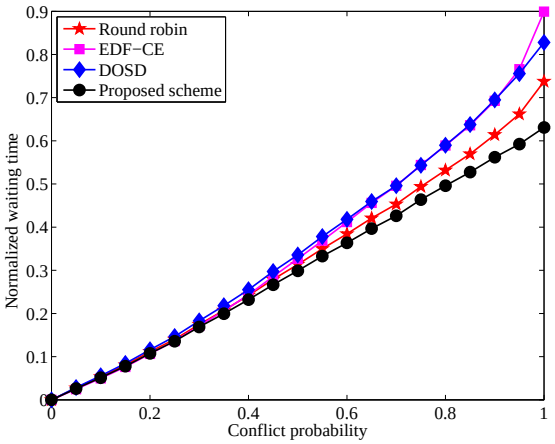

Figure 13: Normalized waiting time for 20 periodic measurement tasks.

Figure 14 shows that the proposed scheduling scheme and the EDF-CE scheme provide similar execution success ratio, which is the highest success ratio as compared to round robin and DOSD schemes. We can see that when the conflict probability is lower than 0.5, the performance of all algorithms is similar, but as the conflict probability increases, the performance differences of the schemes become more pronounced. As an interesting obser-

vation, when the conflict probability is 1, where no more than one job can be executed at a time by any of the schemes, the waiting time and success ratio of the schemes show differences. The low success probability of DOSD is expected as it selects jobs with long execution time first, and the remaining time will then be left to a large number of tasks that may be delayed close to or beyond the end of their periods; therefore, a large number of jobs are missed. In the proposed scheduling algorithm, the degree of a sub-vertex is decided by the length of the execution time of the job, so that scheduling by the ascending order of the degree means that the job with the shortest execution time is scheduled first. This selection can potentially save a larger number of time slots for the subsequent jobs in the waiting queue. Therefore, the performance of this algorithm is also the highest with the conflict probability of 1.0.

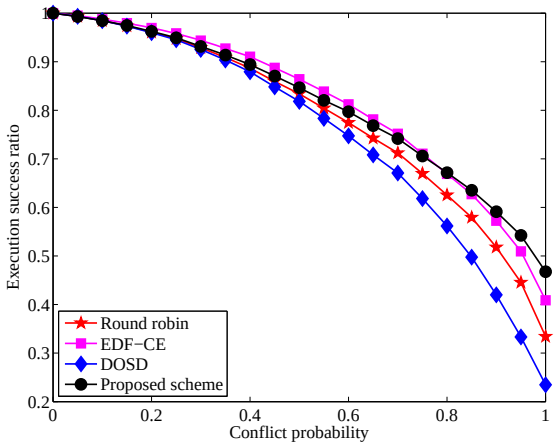

Figure 14: Execution success ratio for 20 periodic measurement tasks.

We also simulated a scenario where the execution times of periodic measurement tasks are non-uniformly distributed. The periodic measurement task set is composed of 10 measurement tasks. The execution time of 5 measurement tasks are uniformly distributed in the range of [2,10] time units while the execution time of the rest of 5 tasks are randomly selected in the range of [8,10] time units. The period of the tasks is uniformly distributed in the range of [11,100] time units. The initial available time of a task is randomly selected in the range of [1,5] time units. The simulation is run 1000 times.

Figure 15 shows the average normalized waiting times under non-uniform distribution in the execution time of the 10 tasks. The large number of tasks with long execution times is not beneficial to the proposed scheme, but the proposed scheme still achieves the lowest normalized waiting time among all compared schemes. The round-robin scheme achieves similar normalized waiting times (although slightly higher) to those of the proposed scheme. The other schemes are favored by this distribution of execution times, but their normalized waiting times are larger than those of the proposed scheme. This indicates that the measurement samples generated by scheduling schemes in comparison are more biased from the regular measurement sampling points, so that the jitter of the time intervals between any two inter-sampling points is large.

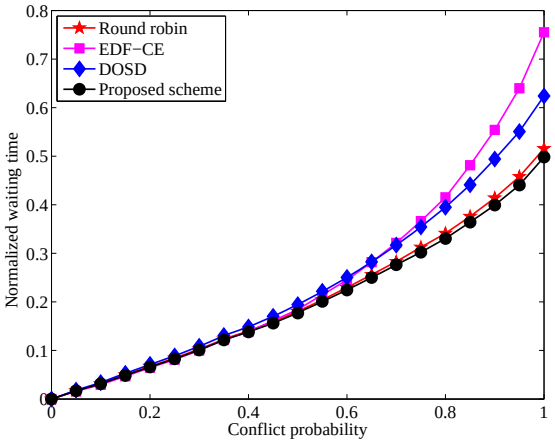

Figure 15: Normalized waiting time for 10 periodic measurement tasks with non-uniformly distributed execution times.

Figure 16 shows the execution success ratios of these schemes for tasks with non-uniformly distributed execution times. The results show that the execution success ratios of all these schemes are lower than the values obtained under execution times with a uniform distribution. The consideration of a larger number of tasks with long execution times makes the scheduling schemes less efficient, and more tasks miss their executions. Nevertheless, the results show that the proposed scheme achieves the highest execution success ratio.

6.4. Simulation Results of On-Demand Tasks Scheduling

We also simulated the scheduling of periodic tasks combined with on-demand tasks and evaluate the performance of the scheme according to the

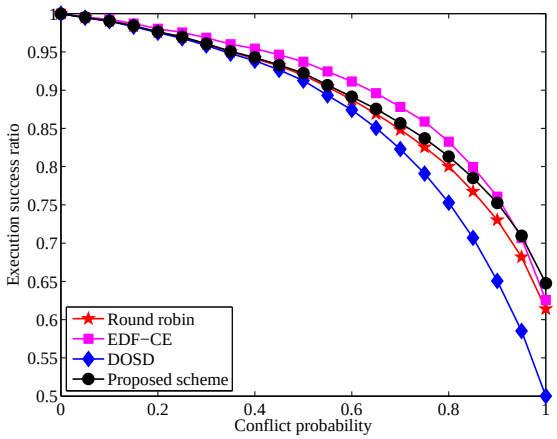

Figure 16: Execution success ratio for 10 periodic measurement tasks with non-uniformly distributed execution times.

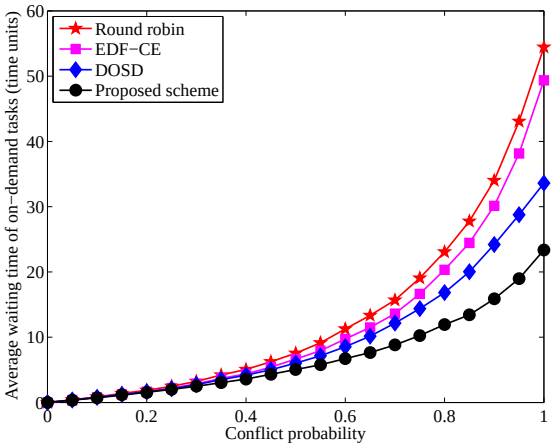

Figure 17: Average waiting time for on-demand measurement tasks of on-demand tasks in a combination with periodic tasks.

average waiting time instead of average normalized waiting time of the jobs since there is no period for the on-demand tasks. In this scenario, there are 10 periodic tasks, and on-demand tasks are created at arbitrary time slots. We combine the periodic tasks with on-demand tasks created at arbitrary time slots, where the arrival of an on-demand measurement task is created with a probability of 0.05 for each time slot. For scheduling (and execution), the priority of on-demand tasks is set to be equal to that of periodic tasks.

The execution and period times are uniformly distributed in the ranges of [2,10] and [11,100] time slots, respectively. As in the previous section, we considered the conflict probability among all measurement tasks (including both periodic and on-demand tasks) increasing from 0 to 1.0 with steps of 0.05. We ran the simulation for 500 times.

Figure 17 shows the average waiting times measured only on the on-demand tasks. The results indicate that the proposed algorithm can achieve the lowest waiting time for on-demand tasks among the considered algorithms as all task are considered with the same priority levels. However, different from the cases with periodic tasks only, the round-robin scheme shows the lowest performance (the longest average waiting time) as some tasks cannot be re-organized with the addition of on-demand tasks because periodic tasks would still follow the pre-determined round-robin order. However, the other schemes follow similar trends as those observed for periodic tasks only.

Figure 18 shows the average waiting times of the periodic tasks only, under this scenario. The results show that the periodic tasks undergo similar average waiting times as in the case of periodic tasks only, and the round-robin scheme and the proposed scheme achieve the lowest average waiting times. The performance of round-robin is high in this scenario as the pre-determined order followed by this scheme isolates the periodic task from the arrivals of on-demand tasks. The proposed scheme, however, accommodates the on-demand tasks and still achieves an efficient outcome, or the lowest average waiting times.

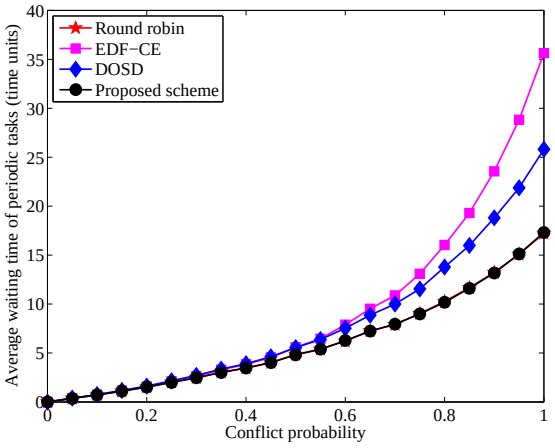

Figure 18: Average waiting time of periodic tasks when they are combined with on-demand tasks.

Figure 19 shows the normalized waiting times of the periodic measurement tasks. This graph

also corroborates the previous observations, where the periodic tasks have similar results to the case of only periodic tasks, with the proposed scheme achieving the highest performance and the EDF-CE scheme achieving the lowest performance.

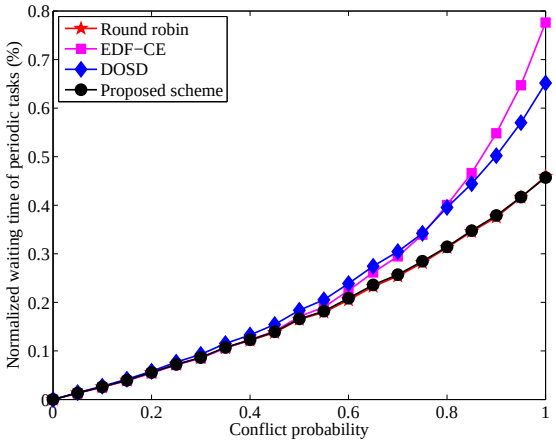

Figure 19: Normalized waiting time of periodic tasks when they are combined with on-demand tasks.

7. Conclusions

In this paper, we have analyzed the problem of contention for resources in network active measurement. The scheduling of active measurement tasks can be used to resolve this contention to allow high utilization of network resources and to provide accurate measurement results and least disturbance to users' data traffic. Critical contentions for any resource are defined as conflicts.

Based on graph coloring theory, we have proposed to describe the measurement tasks relation by using a conflict graph, and to convert this scheduling problem into a graph coloring problem. We have also proposed two algorithms to schedule tasks according to the ascending order of the degree of sub-vertices in the conflict graph, one for periodic measurement tasks, and another for on-demand measurement tasks. Each sub-vertex represents one basic time unit for the execution time of the task. The results showed that the proposed scheduling schemes provide the shortest average waiting time for cases where periodic tasks are considered in the network as well as when on-demand task are added in a network with existing periodic tasks. The proposed schemes also achieve the highest utilization of network resources as shown by achieving the highest execution success ratios in the presented results.

In addition, the schemes are able to schedule the on-demand tasks with either higher or equal priority with respect to that of the periodic measurement tasks.

Acknowledgement

This work has been partially supported by Computation and Communication: Promoting Research Integration in Science and Mathematics (C2PRISM), NSF GK-12 Project #0638423, at New Jersey Institute of Technology.

References

[1] L. Ciavattone, A. Morton, and G. Ramachandran, "Standardized active measurements on a tier 1 IP backbone," *IEEE Communications Magazine*, Vol. 41, Iss. 6, pp. 90-97, Jun. 2003.

[2] T. Tsugawa, T.M. Cao-Leh, G. Hasegawa, and M. Murata, "Inline bandwidth measurements: Implementation difficulties and their solutions," *IEEE Workshop on End-to-End Monitoring Techniques and Services (E2EMON)*, pp. 1-8, May 2007.

[3] G. Dos Santos *et al.*, "UAMA: a unified architecture for active measurements in IP networks; End-to-end objetive quality indicators," *IEEE/IFIP Integrated Network Management Symposium*, pp. 246-253, May 2007.

[4] M. Zhanikeev, S. Xu, and Y. Tanaka, "Active performance measurement for IP over all-optical networks," *IEEE/IFIP International Conference in Central Asia on Internet*, pp. 1-5, Sep. 2006.

[5] M. Zhanikeev and Y. Tanaka, "A testbed for agent-based multi-purpose extensible active measurement," *Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM)*, Mar. 2006.

[6] M. Mushtaq and T. Ahmed, "Adaptive packet video streaming over P2P networks using active measurements," *IEEE Symposium on Computers and Communications (ISCC)*, pp. 423-428, Jun. 2006.

[7] R. Mishra and V. Sharma, "QoS routing in MPLS networks using active measurements," *IEEE Conference on Convergent Technologies for Asia-Pacific Region (TENCON)*, pp. 323-327, Oct. 2003.

[8] M. Zangrilli and B. Lowekamp, "Comparing passive network monitoring of grid application traffic with active probes," *Grid Computing Workshop*, pp. 84-91, Nov. 2003.

[9] M. Aida, K. Ishibashi, and T. Kanazawa, "CoMPACT-Monitor: change-of-measure based passive/active monitoring weighted active sampling scheme to infer QoS," *Applications and the Internet (SAINT) Workshops*, pp. 119-125, Feb. 2002.

[10] P. Calyam, D. Krymskiy, M. Sridharan and P. Schopis, "Active and passive measurements on campus, regional and national network backbone paths," *IEEE Conference on Computer Communications and Networks (ICCCN)*, pp. 537-542, Oct. 2005.

[11] V. Sharma and M. Suma, "Estimating traffic parameters in Internet via active measurements for QoS and congestion control," *IEEE Global Telecommunications Conference (GLOBECOM)*, pp. 2527-2531, Nov. 2001.

[12] K. Mase and Y. Toyama, "End-to-end measurement based admission control for VoIP networks," *IEEE Communications Conference (ICC)*, pp. 1194-1198, Apr. 2002.

[13] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," *IEEE Network*, Vol. 17, Iss. 6, pp. 27-35, Nov.-Dec. 2003.

[14] M. Luckie and A. McGregor, "IPMP: IP measurement protocol," *in Proc. of Passive and Active Measurement Workshop*, pp. 168-176, Apr. 2002.

[15] S. Shalunov and B. Teittelbaum, "One-way active measurement protocol (OWAMP)," in *IETF*, RFC3763, 2004.

[16] Z. Qin, R. Rojas-Cessa, and N. Ansari, "Distributed link-state measurement for QoS routing," *IEEE Military Communications Conference (MILCOM)*, pp. 1-6, Oct. 2006.

[17] J. Sommers and P. Barford, "An active measurement system for shared environments," *ACM SIGCOMM Conference on Internet Measurement (IMC)*, pp. 303-314, Oct. 2007.

[18] E2E piPEs, [online] available: `http://e2epi.internet2.edu/e2epipes/`

[19] N. Nu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," *IEEE JSAC*, Vol. 21, No. 6, pp. 879-894, Aug. 2003.

[20] perfSONAR, [online] available: `http://www.perfsonar.net/`

[21] Y. Labit, P. Owezarski, and N. Larrieu, "Evaluation of active measurement tools for bandwidth estimation in real environment," *End-to-End Monitoring Techniques and Services Workshop*, pp. 71-85, May 2005.

[22] National laboratory for Applied Network Research (NLANR), Active Measurement Project (AMP), [online] available: `http://watt.nlanr.net/`

[23] F. Strohmeier, H. Dorken, and B. Hechenleitner, "Aquila Distributed QoS Measurement," Aquila Project, Aug. 2001.

[24] Pipechar, [online] available: `http://dsd.lbl.gov/NCS/`

[25] Pathload, [online] available: `http://www-static.cc.gatech.edu/fac/Constantinos.Dovrolis/pathload.html`

[26] K. Anagnostakis, M. Greenwald, and R. Ryger, "Cing: measuring network-internal delays using only existing infrastructure," *IEEE Conference on Computer Communications (INFOCOM)*, pp. 2112-2121, Mar. 2006.

[27] A. Downey, "Clink: a tool for estimating Internet link characteristics," [online] available: `http://allendowney.com /research/clink/`

[28] K. Lai and M. Baker, "Nettimer: a tool for measuring bottleneck link bandwidth," *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Mar. 2001.

[29] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?" *IEEE Conference on Computer Communications (INFOCOM)*, pp. 905-914, Apr. 2001.

[30] Pathchar, [Online] available: `http://www.caida.org/tools/utilities/others/pathchar/`

[31] J. Sommers, P. Barford, and W. Willinger, "Laboratory-based calibration of available bandwidth estimation tools," *Microprocess. Microsyst.*, Vol. 31, Iss. 4, pp. 222-235, Jun. 2007.

[32] "One-way Ping (OWAMP)," [online] available: `http://e2epi.internet2.edu/owamp/`

[33] R. Graham, E. Lawler, J. Lenstra and A. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Annals of Discrete Mathematics*, pp. 5:287-326, 1979.

[34] T. McGregor, H. Braun, and J. Brown, "The NLAMR network analysis infrastructure," *IEEE Communications Magazine*, Vol. 38, Iss. 5, pp. 122-128, May 2000.

[35] S. Kalidindi and M. Zekauskas, "Surveyor: an infrastructure for internet performance measurements," *Internet Global Summit(INET)*, Jun. 1999. [Online] available: `http://www. isoc.org/ inet99/proceedings/4h/4h_2.htm`

[36] R. Wolski, N. Spring, and C. Peterson, "Implementing a performance forecasting system for metacomputing: The network weather service," *In Proceedings of Supercomputing '97*, Aug. 1997.

[37] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, Vol. 20, No. 1, pp. 46-61, Jan. 1973.

[38] K. Jeffay, D. Stanat, and C. Martel, "On nonpreemptive scheduling of periodic and sporadic tasks," *IEEE Real-Time Systems Symposium*, pp. 129-139, Dec. 1991.

[39] Y. Cai and M. Kong, "Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems," *Algorithmica*, Vol. 15, No. 6, pp. 15: 572-599, Jun. 1996.

[40] P. Calyam, C. Lee, P. Arava, and D. Krymskiy, "Enhanced EDF scheduling algorithms for orchestrating network-wide active measurements," *IEEE Real-Time Systems Symposium (RTSS)*, 10 pages, Dec. 2005.

[41] I. Gopal, M. Bonuccelli, and C. Wong, "Scheduling in multibeam satellites with interfering zones," *IEEE Transactions on Communications*, Vol. 31, Iss. 8, pp. 941-951, Aug. 1983.

[42] W. Chen, P. Sheu, and J. Yu, "Time slot assignment in TDM multicast switching systems," *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1296-1305, Apr. 1991.

[43] A. Bagchi and S. Hakimi, "Data transfers in broadcast networks," *IEEE Transactions on Computers*, Vol. 41, Iss. 7, pp. 842-847, Jul. 1992.

[44] J. Goossens and C. Macq, "Limitation of the hyperperiod in real-time periodic task set generation," *In Proceedings of the RTS Embedded System (RTS'01)*, pp. 133-147, 2001.