# Scheduling Tasks with Markov-Chain Based Constraints

Donglin Liu, Xiaobo Sharon Hu
Dept. of Computer Science and Engineering
Univ. of Notre Dame, IN 46556, USA
dliu, shu@nd.edu

Michael D. Lemmon, Qiang Ling
Dept. of Electrical Engineering
Univ. of Notre Dame, IN 46556, USA
lemmon, qling@nd.edu

## Abstract

*Markov-Chain (MC) based constraints have been shown to be an effective QoS measure for a class of real-time systems, particularly those arising from control applications. Scheduling tasks with MC constraints introduces new challenges because these constraints require not only specific task finishing patterns but also certain task completion probability. Multiple tasks with different MC constraints competing for the same resource further complicates the problem. In this paper, we study the problem of scheduling multiple tasks with different MC constraints. We present two scheduling approaches which $(i)$ lead to improvements in "overall" system performance, and $(ii)$ allow the system to achieve graceful degradation as system load increases. The two scheduling approaches differ in their complexities and performances. We have implemented our scheduling algorithms in the QNX real-time operating system environment and used the setup for several realistic control tasks. Data collected from the experiments as well as simulation all show that our new scheduling algorithms outperform algorithms designed for window-based constraints as well as previous algorithms designed for handling MC constraints.*

## 1 Introduction

In many real-time applications such as networked control systems and multimedia services, jobs are allowed to occasionally miss their deadlines and such jobs are often discarded or dropped since finishing these late jobs does not improve system performance. Such systems are referred to as Firm real-time systems (FRTS) [6]. Quantifying the "occasional deadline misses" is critical for evaluating the Quality of Service (QoS) of a FRTS. One group of often used QoS metrics for FRTS are based on the average behavior of a task's long execution sequence such as deadline miss ratio or average dropout rate, effective processor utilization and completion count [4, 5]. Another group of QoS metrics for FRTS are based on the task's completion pattern within a certain "window" [9, 20, 22, 6, 11].

In [16], Liu, et. al. introduced a new type of constraints for FRTS, i.e., Markov Chain (MC) based constraints. An MC constraint specifies a task's desired dropout process in the form of a Markov Chain. That is, the probability of discarding or dropping a job (i.e., a task's instance) depends on the current state of the task dropout process. MC constraint bear similarities with the average behavior based constraints as well as the window based constraints, but it is more general than both. In fact, these two types of constraints are special cases of MC constraints. More importantly, it is shown in [16] that MC constraints provide the flexibility and preciseness needed to specify the desired stochastic behavior of a system. This is particularly valuable for those FRTS whose QoS can be expressed as a function of the state transition probabilities in the associated dropout process, which is the case for many control applications [15].

In this paper, we consider FRTS containing multiple tasks with different MC constraints. Such FRTS can be found in a number of applications, e.g., a robot control system which must execute several control tasks on one resource. In such a system, the performance of each control task is a function of the respective dropout process and the control tasks need to be coordinated to accomplish some common goal. Existing scheduling algorithms, as we will show later, are not able to achieve good overall performance due to their various drawbacks.

We present two new scheduling approaches, the State Sensitivity based algorithm (referred to as SSA) and Grouped Fair Dropout Rate algorithm (referred to as GFDR), for the FRTS under consideration. Both SSA and GFDR partition jobs into three priority groups, instead of two as done by most FRTS schedulers. They differ in their methods of assigning priorities to jobs within each priority group. In SSA, we adopt a novel priority assignment scheme based on some interesting observations. GFDR used a more straightforward intra-group priority assignment scheme which trades in quality for efficiency. Both algorithms effectively overcome the shortcomings of the existing algorithms as shown by simulation results.

For new scheduling algorithms, it is important to study their effect in a real-world scenario. Towards this end, we have implemented our new scheduling approaches as well as existing ones pertinent to this work in the QNX real-time operating system environment [13]. The observations of the dropout processes in this experimental setup again demonstrate the superiority of the new scheduling algorithms.

The remainder of this paper is organized as follows. Section 2 provides preliminary information. In Section 3, we discuss some useful observations about MC constraints and our proposed methods to solve the scheduling problem. Section 4 presents the detailed experimental setup as well as experimental data. We conclude the paper in Section 5.

## 2 Preliminaries, motivation and related work

In this section, we first review the MC constraint concept. (Interested readers are referred to [16] for more details.) Then we use an example to motivate the problem to be solved and give a formal problem formulation. We last discuss related work.

### 2.1 MC constraints

Before presenting the MC constraint definition, we introduce some necessary notation. Let $\tau_i$ be the $i$-th task, and $\tau_{ij}$ be the $j$th instance or job of $\tau_i$. The period of $\tau_i$ is denoted by $t_i$, and the deadline of $\tau_i$ is denoted by $d_i$. Without loss of generality, we assume that the execution time of $\tau_i$ is a random variable denoted by $C_i$. We use $f_{ij}$ to denote the completion status of job $\tau_{ij}$, i.e., $f_{ij} = 1$ if $\tau_{ij}$ is completed at or before its deadline and $f_{ij} = 0$ otherwise. Note that $f_{ij} = 0$ is due to either the late finish of $\tau_{ij}$ or the rejection of $\tau_{ij}$ before it is started. For simplicity, we refer to both cases as the jobs being dropped. The average dropout rate of task $\tau_i$ is denoted by $\theta_i$.

Given a task $\tau_i$, we refer to the completion status of a long sequence of $\tau_i$'s jobs as its dropout process and denote it by $\mathcal{DP}_i$. Each $\mathcal{DP}_i$ can be considered as a discrete stochastic process since the task execution time is a random variable. An MC constraint is defined for a task's dropout process to indicate that the dropout process should follow the MC constraint. An MC constraint for task $\tau_i$ is denoted by $\mathcal{MC}_i$. $\mathcal{MC}_i$ is a discrete stochastic process with $M_i$ states and $M_i \geq 2$. Each state is represented by $X^i_m$ where $m = 1, 2 \cdots, M_i$. The transition probability from one state $X^i_j$ to another $X^i_k$ indicates either the probability of the next job being dropped or the probability of the next job being completed. Each state is associated with the dropout pattern expressed as a binary string of the status bits $f_{ij}$ to represent the most recent completion status for $\tau_{ij}$. For example, if two bits are used for each state in $\mathcal{MC}_i$, then there are at most 4 states ($M_i = 4$) in $\mathcal{MC}_i$. The state which represents the execution pattern of one completed job followed by one dropped job can be written as $X^i_m = \{10\}$, and so on. Figure 1(a) depicts the general MC process for the case of using two completion status bits for each state, where $\epsilon_1$ to $\epsilon_4$ are the probabilities of the next job being dropped at the respective states. Given the fact that a state can only transition to at most two other states based on the completion status of the next job, the dropout probability (i.e., dropout rate) at each state, denoted by $\epsilon_m$, for $m = 1, 2, \cdots, M_i$, is sufficient to uniquely specify $\mathcal{MC}_i$. Note that the average dropout rate of task $\tau_i$, $\theta_i$, is a function of the dropout probability at each state. Figure 1(b) shows a specific example of an MC constraint, which has three states and $\epsilon_1$ represents the dropout probability at state $X_{(1)} = \{11\}$. It is not difficult to verify that this MC constraint is equivalent to the $(m, k)$-*firm* constraint [9], where $m = 2$ and $k = 3$. Figure 1(c) illustrates another MC constraint which cannot be described by any window based constraint.

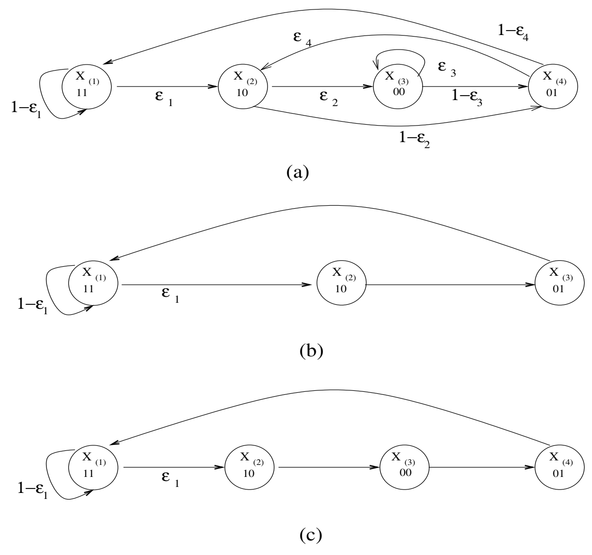


**Figure 1.** (a) A general MC dropout process with 2 bits for each state. (b) An example MC constraint, which is equivalent to $(2, 3)$-*firm* constraint. (c) An example MC constraint, which describes the optimal dropout process in a networked control system.

### 2.2 A motivational example

From the discussion of MC constraints, it is not difficult to see that MC constraints generalize not only average behavior based constraints but also window based constraints. Another unique feature of MC constraints is that they can be directly related to the QoS of a task. We use a networked control system (NCS) to illustrate this and motivate the needs for MC constraints. The example will also be used in later sections for explaining our algorithms.

Networked control systems are widely used in modern control applications. An NCS is a feedback control system where sensor and actuator data are sent to and from the control plant over a communication network. These data are occasionally delayed due to one of two reasons: either the medium is unreliable or the network is overloaded. Since a delayed signal is useless to the system, an NCS can be categorized as a FRTS. In [15], Ling and Lemmon show that an NCS's performance can be expressed as a function of the dropout process (represented by an Markov chain) of the communication network. To optimize the performance of an NCS, the dropout process should follow a specific MC process. This MC process can thus be considered as the constraint for the dropout process.

In Figure 2 (top), a typical NCS model is given. This system consists of a loop function $L(z)$ whose input, $w[n]$, is a zero-mean white noise disturbance. The output $y[n]$ is fed back through the network in a manner that is controlled by a *dropout process*, $\{d[n]\}$. The dropout process is a binary random process that models the behavior of the network. Since this particular control system tries to reject disturbances, the system's performance, i.e., QoS, is characterized by the output signal power. Varying the dropout process even for the same average dropout rate can lead to different system performance. Ling and Lemmon show that the dropout process that maximizes the system performance (i.e., minimizes the output signal power) follows the Markov Chain given in Figure 1(c) [15].

To better understand the implications of MC constraints, we reproduce a figure from [15] in Figure 2(bottom). This figure depicts three curves corresponding to the system output signal power when the system dropout process follows three different distributions, i.e., the one given in Figure 1(b), an independent and identically distributed process, and the one given in Figure 1(c). For a given average dropout rate (the x-axis), different dropout processes lead to different output signal power values. Since lower output signal power is more desirable, it is clear that following the Markov Chain in Figure 1(c), i.e., the MC constraint for the system, results in optimal system performance. This shows that the system QoS depends on not only the average dropout rate but also the dropout patterns. Furthermore, this QoS can be expressed as a function of the dropout process and the QoS is maximized when the dropout process satisfies the MC constraint at every average dropout rate value.

The above example uses a communication network to illustrate signal dropouts. However, this can be generalized to any FRTS in which multiple "consumers" compete for a limited resource. For example, in a robot control system a CPU may need to receive data from multiple sensors, execute several control tasks, and send data to a number of actuators. All of these tasks compete for the CPU resource and some jobs may have to be dropped when the CPU is overloaded. How to design a scheduler to improve the overall system performance can be very challenging.

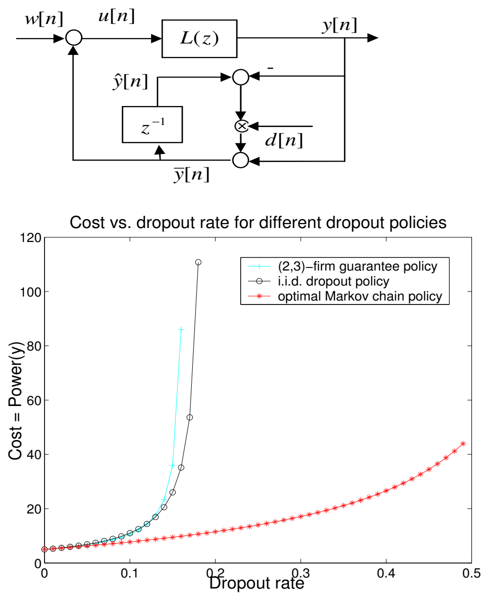


**Figure 2.** Top - NCS model: Bottom - comparison of the three dropout models

## 2.3 Problem Formulation and Related Work

We are interested in solving the following problem:

**Problem 1** Given an FRTS with $n$ tasks, $\tau_1, \tau_2, \ldots, \tau_n$, which share a single resource, and each task $\tau_i$ is associated with an MC constraint $\mathcal{MC}_i$, design a scheduling approach for the resource to maximize overall system performance.

We assume that the system utilization is larger than one in the worst case, otherwise the problem is trivial when the earliest deadline first (EDF) scheduling algorithm is used. This problem bears some similarity with the resource allocation problem for the Q-RAM model [12, 18, 19] since both strive to quantitatively measure QoS. One might consider the dropout process of each task as a QoS dimension. Then optimizing the overall system performance can be treated as maximizing the system utility defined as a function of the task dropout process [12]. A key difference between the problem under consideration and the Q-RAM model is that the QoS measure of the former is a stochastic process while the QoS measures for the latter are deterministic. Furthermore, the resource allocation approach for Q-RAM can be quite costly if used online.

Many papers have studied the scheduling problem for overloaded systems with probabilistic execution times. Tia *et. al.* uses a task transformation method to separate jobs into different groups which are executed either by a rate-monotonic scheduler (RMS) or a sporadic server [21]. Atlas and Bestavros propose the stochastic rate-monotonic scheduling approach which employs a novel admission control strategy to ensure task isolation and job schedulability [2]. Admission control based on control theory and other methods such as slack stealing have also been discussed (e.g., [3, 17]). All of these methods are only concerned with task completion probability as the QoS. This is the same as considering only the average dropout rate while ignoring the underlying dropout process in our problem. As we have shown earlier, same average dropout rate may not necessarily lead to the same performance (see Figure 2). Scheduling algorithms for systems with MC constraints must pay close attention to job dropout patterns.

Scheduling approaches for systems with window-based constraints indeed consider dropout patterns (e.g., [8, 10, 11, 9, 20, 22]). These papers address overload situations by partitioning jobs into two priority groups: the mandatory and optional. An underlying assumption of such partitioning is that the system performance improves as more optional jobs are finished on time. For a task with an MC constraint, this assumption is no longer true. For example, observe the optimal Markov Chain in Figure 1 for the system in Figure 2(top), it is desirable to actually drop the next job if the dropout process is at state $\{10\}$.

There are several unique challenges in designing scheduler for tasks with MC constraints. First, the scheduling goal for a task with an MC constraint is to achieve a dropout process as similar as possible to the MC constraint. Recall that a dropout process is a stochastic process and is best evaluated by relatively long history dropout pattern. However, when designing an online scheduler, the scheduler can only afford to make decisions based on "local" information such as temporary system load. Second, merely forcing the dropout process of a task to follow the given MC constraints without any concern for the average dropout rate could still hurt the task performance. This can be seen from Figure 2 (bottom) where the output signal power can be less desirable if the dropout process follows the optimal Markov Chain at a much higher average dropout rate (say $45\%$) than if it follows the i.i.d. process at a lower average dropout rate (say $12\%$). Thus, the scheduler must "fairly" allocate the resource. Third, depending on the current state of the dropout process of each task, finishing on time the next job of a certain task can be much more important than that of another task. It is desirable to have a strategy for evaluating the criticality of each state.

From the above discussion, it can be seen that any algorithms that target at only average dropout behavior or window based patterns cannot adequately address the challenges introduced by MC constraints. Liu, et. al. proposed three scheduling algorithms (MDA, DDA, and FDA) which consider specifically MC constraints. These algorithms determine online whether a job of a task with an MC constraint should be a must-finish job or an optional-finish job. Jobs within each group are scheduled by the EDF algorithm. Algorithms in [16] addressed the first challenge discussed above. However, they do not explicitly address the second and third challenge since there is no consideration of "fairness" or "criticality". Furthermore, there is no strategy for comparing tasks with different MC constraints, and the evaluation of the overall system performance is not formal. The work presented in this paper intends to overcome these shortcomings.

## 3 Scheduling tasks with MC constraints

To find a good scheduling strategy for maximizing the overall performance of a system with multiple tasks with different MC constraints, we need to quantify *overall system performance*. Since each task is associated with an MC constraint, it is natural for each task to express its performance as a function of the respective dropout process as well as the average dropout rate. For example, for the task model in Figure 2, its performance can be considered as the reciprocal of the output signal power which depends on both the dropout process and the average dropout rate (see Figure 2 (bottom)).

Let $\mathcal{P}_i = G_i(\mathcal{DP}_i, \theta_i)$ denote the performance of $\tau_i$ where $G_i()$ represents a function. It is not difficult to see that $\mathcal{P}_i$ has a similar meaning as the utility function used by the Q-RAM model [12]. (We choose to use performance instead of utility since utility might be miss interpreted as some measure of system utilization.) Therefore, we employ the same way as the Q-RAM model to define the overall system performance. Let **P** denote the overall system performance. Then, **P** is obtained by

$$\begin{aligned}\mathbf{P} &= \sum_i^n w_i \overline{\mathcal{P}}_i \\ &= \sum_i^n \frac{w_i(G_i(\mathcal{DP}_i, \theta_i) - \min\limits_{\theta_i} G_i(\mathcal{DP}_i, \theta_i))}{\max\limits_{\theta_i} G_i(\mathcal{DP}_i, \theta_i) - \min\limits_{\theta_i} G_i(\mathcal{DP}_i, \theta_i)}, \qquad (1)\end{aligned}$$

where $w_i$ is the weight or relative importance of $\tau_i$, and $\overline{\mathcal{P}}_i$ is the normalized performance of $\tau_i$. Evaluating $\mathcal{P}_i$ can be computationally intensive since it involves statistical analysis of a long job dropout pattern (such as over one million jobs). This can be tolerated if the analysis is done offline. Evaluating **P** online for scheduling decisions is too costly.

In SSA, we use a hybrid offline/online scheduling approach. A priority driven, preemptive scheduler is used online to dispatch jobs. Due to the dynamic nature of a task dropout process, dynamically assigned priorities are preferred for meeting MC constraints. Our challenge then is how to determine the priorities of jobs upon their release. We approach the online priority assignment problem by carefully studying the properties of MC constraints. Given each possible state of the dropout process, we want to offline predict what types of future dropout patterns would be more desirable or more hurtful in terms of overall performance. This prediction is used online to guide the priority assignment process so as to enforce certain types of future dropout patterns. Unlike SSA, GFDR is a purely online algorithm which used a more straightforward heuristic approach. We will discuss our approaches in detail below.

## 3.1 Observations and implications

**Observation 1** *There are at most three types of states in any MC constraint:*

1. **Must-Finish (MF) state:***state from which the dropout probability of the next job is equal to* 0
2. **Better-Finish (BF) state:***state from which the dropout probability of the next job is between* 0 *and* 1
3. **Better-Drop (BD) state:** *state from which the dropout probability of the next job is equal to* 1

For example, state $\{10\}$ in Figure 1(b), state $\{11\}$ in Figure 1(b) and state $\{10\}$ in Figure 1(c) are MF, BF and BD states respectively. Given task $\tau_i$ and $\mathcal{MC}_i$, by monitoring the most recent job finishing patterns, the state of $\mathcal{DP}_i$ (the dropout process of $\tau_i$) at each job release as well as the type of the state can be readily determined. For example, if a two-bit string is used in $\mathcal{MC}_i$, the state of $\mathcal{DP}_i$ at the release of $\tau_{ij}$ is simply $\{f_{i(j-2)}f_{i(j-1)}\}$. For simplicity, we refer to the state of $\mathcal{DP}_i$ at the release of $\tau_{ij}$ as the state which $\tau_{ij}$ is in or just $\tau_{ij}$'s state.

Based on Observation 1, a scheduler should strive to treat jobs in different types of states differently so as to follow a given MC constraint. Specifically, we introduce three priority groups for jobs with MC constraints, i.e., MF group, BF group, and BD group in the order of decreasing priority levels. Jobs in an MF state are assigned to the MF group since a miss in an MF state would violate the basic Markov Chain transition structure. The jobs in a BF state are put to the BF group as a temporary miss at this state is acceptable. In other words, a miss in an MF state causes the system performance to degrade much more severely than a miss in a BF state. Jobs in a BD state are assigned to the BD group such that they are only executed if there is no any other resource demand. We do not simply drop jobs at a BD state since finishing them could improve the system performance in a long run by decreasing the average dropout rate. Note that this 3-way partitioning method is an extension of the mandatory v.s. optional partitioning used by existing work.

There can be more than one job in each priority group. To prioritize the jobs within a group, one might consider to use EDF or RM for simplicity or other algorithms aimed at maximizing the completion ratio (e.g., [5]). However, as we pointed out earlier, these algorithms do not address the special requirements of MC constraints. As system load increasing, some jobs in the BF or even MF group may be dropped. Dropping different jobs can degrade the system performance differently. Ideally, we would like to drop the jobs which have the least negative "impact" on system performance as defined in equation (1). The more negative the impact is, the higher priority this job should receive. Such impact depends on both $w_i$ and $G_i$ in equation (1). The weight of each task, $w_i$, assuming known, reflects the relative importance of the task to the overall application, which is out of the scope of this paper. More illusive is to capture the impact of the task dropout process, $\mathcal{DP}_i$, at the release of $\tau_{ij}$ on the performance. We have pointed out earlier that evaluating function $G_i$ online is computationally prohibitive.

The following observation and subsequent discussion help shed some light on how to quantify the impact of dropping a job.

**Observation 2** *The impact of a job's completion status (miss or meet its deadline) on the system performance depends on the job's current state.*

The observation is quite intuitive since the completion status bits of a long sequence of jobs determine the task dropout process which in turn determines the task performance. If one could offline assess for every task the impact of missing/meeting a job's deadline at each state on the performance, this assessment together with the relative importance of the tasks can then be used online to prioritize jobs from different tasks. However, the task performance is a function of the task dropout process which depends on long job-execution patterns. It is not immediately apparent how to evaluate the changes in system performance for missing or meeting a single job deadline at a particular state, since this is a transient behavior. We will show how we tackle this problem in the next subsection.

## 3.2 State Sensitivity

Since it is difficult to quantify transient behavior in a dropout process, an alternative is to model the effect of transient behavior statistically. For example, consider a task's dropout process, $\mathcal{DP}_i$, is at a specific state, say $X^i_m$. If we let a miss from $X^i_m$ occur at some probability, the performance of the resulting dropout process can be evaluated either analytically [15] or through statistical simulation depending on the actual application. (This is just one step in the process of finding the optimal dropout process.) However, having the performance of a dropout process is not sufficient. To evaluate the impact discussed in the last subsection, we need to find the performance *change* with respect to some deadline miss or meet.

Recall that MC constraint $\mathcal{MC}_i$ represents the most desirable dropout process. Deviations from $\mathcal{MC}_i$ is the cause for the performance to degrade. Therefore, changes in performance could be measured by comparing the performance of some dropout process with that of $\mathcal{MC}_i$. We are now ready to introduce the concept of state sensitivity.

**Definition 1 State sensitivity:** *For a given task $\tau_i$ and $\mathcal{MC}_i$, the state sensitivity $\eta$ at state $X_m^i \in \mathcal{MC}_i$ is*

$$\eta(X_m^i, \theta_i, \tilde{\epsilon}_{im}) = \frac{G_i(\mathcal{DP}_i^*, \theta_i^*) - G_i(\mathcal{MC}_i, \theta_i)}{|\theta_i^* - \theta_i|}. \quad (2)$$

*$\theta_i$ is the average dropout rate of $\tau_i$, $\tilde{\epsilon}_{im}$ is the* deviation *of the dropout probability for some dropout process at state $X_m^i$ from that given in $\mathcal{MC}_i$, $\mathcal{DP}_i^*$ is the dropout process induced by $\tilde{\epsilon}_{im}$ (i.e., $\epsilon_{im}^* = \tilde{\epsilon}_{im} + \epsilon_{im}$, where $\epsilon_{im}^*$ and $\epsilon_{im}$ are the dropout probabilities of $\mathcal{DP}_i^*$ and $\mathcal{MC}_i$, respectively), and $\theta_i^*$ is the average dropout rate of $\mathcal{DP}_i^*$.*

In other words, sensitivity at a certain state defined in $\mathcal{MC}_i$ captures the performance change of a dropout process $\mathcal{DP}_i^*$ from the optimal dropout process $\mathcal{MC}_i$. State sensitivity measures this change (caused by a perturbation in the dropout probability at a state) with respect to the change in resource utilization indicated by average dropout rate.

According to the above definition, one can see that state sensitivity reflects the impact on performance when the dropout process of $\tau_i$ deviates from $\mathcal{MC}_i$. A larger sensitivity value at a given state means that the performance would degrade more severely as the probability of missing a job's deadline at this state becomes larger than that given in $\mathcal{MC}_i$. Therefore, the job should be given a higher priority. It follows then that the state sensitivity values together with task weights can be used online to prioritize jobs in the same priority group.

To compute $\eta(X_m^i, \theta_i, \tilde{\epsilon}_{im})$ offline, the key is to evaluate $G_i(\mathcal{DP}_i^*, \theta_i^*)$ for given $\tilde{\epsilon}_{im}$ and $\theta_i$ values, since $G_i(\mathcal{MC}_i, \theta_i)$ should already be available for a given $\mathcal{MC}_i$. Based on the discussion at the beginning of this subsection, we propose to evaluate $G_i(\mathcal{DP}_i^*, \theta_i^*)$ as follows. For any $\theta_i$, we first determine one set of transition probabilities in $\mathcal{MC}_i$ that satisfy $\theta_i$. (If there exist more than one such set of probability values, we can take any set since they all result in the same performance.) Now, for each state $X_m^i$ in $\mathcal{MC}_i$, we perturb $\mathcal{MC}_i$ by allowing the two transition probabilities at state $X_m^i$ to change according to $\tilde{\epsilon}_{im}$ while keeping the rest of the transition probabilities fixed. The resulted dropout probability at state $X_m^i$ is $\epsilon_{im}^*$. For each value of $\epsilon_{im}^*$, $\theta_i^*$ of $\mathcal{DP}_i^*$ can be easily computed. Then, the performance of the resulting dropout process, i.e., $G_i(\mathcal{DP}_i^*, \theta_i^*)$, can be evaluated and $\eta(X_m^i, \theta_i, \tilde{\epsilon}_{im})$ can be readily obtained.

We use the MC constraint in Figure 1(c) to illustrate how the above process works. We first select the initial average dropout rate $\theta_i$ to be 25%. Based on the chosen average dropout rate of 25%, we have $\epsilon_1 = 20\%$. Take state $\{01\}$ in Figure 1(c) as an example. We now modify the Markov Chain by setting the transition probability from state $\{01\}$ to $\{10\}$ to be $\epsilon_4$ (the probability of of a job being

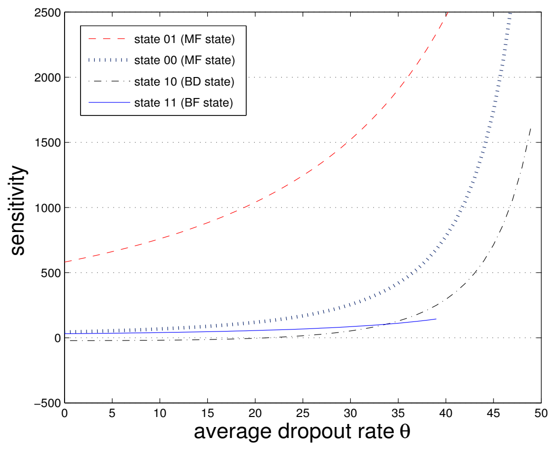


**Figure 3.** State sensitivity for every state in Figure 1(c).

dropped), and that to state $\{11\}$ to be $1 - \epsilon_4$ and leaving others unaltered. For each selected initial average dropout value, we have a corresponding dropout process. The system performance of this dropout process is then evaluated by simulating in `Matlab` the control model given in Figure 2(top) [15].

The state sensitivity values v.s. the average dropout rate for every state in Figure 1(c) while setting $\tilde{\epsilon}_{im} = 0.1$ are plotted in Figure 3. The plots clearly indicate that for the same initial average dropout rate, different states in the MF group have different sensitivity values, which can be used for priority assignments. The curve for BD state deserves special consideration. When $\theta_i$ is less than 16%, the sensitivity of the BD state is a very small negative value. When $\theta_i$ is larger than 35%, the sensitivity of the BD state is larger than that of the BF state. Note that dropout probability of a BD state in MC constraint is equal to 1. Unlike the sensitivity of other states, the sensitivity value for a BD state shows performance changes when more jobs in BD state *meet their deadlines*. Thus the curve of BD state indicates that we should try to finish jobs in BD states only when there are no jobs from other groups.

One might argue that perhaps sensitivity alone is sufficient for priority assignment. However, when comparing states from different MC constraints, the sensitivity of one type of states from one MC constraint is of no relevance to that of another type of states from another MC constraint. Thus, we need both group partitioning and sensitivity-based prioritizing. A careful reader may notice that it is possible for a dropout process to be at a state not included in the MC constraint. In such case, the scheduler should either assign the highest priority to or drop the very next job depending on which action will make the dropout process go to one of states in $\mathcal{MC}_i$.

## 3.3 State-sensitivity based algorithm

We are now ready to summarize our state-sensitivity based scheduling approach. SSA consists of two parts: the offline part (SSA-Offline) and the online part (SSA-online). We summarize the two parts in Algorithm 1 and 2.

**Algorithm 1** SSA-offline

```
Input: Task set $\mathcal{T}$, for each $\tau_i \in \mathcal{T}$: $\mathcal{MC}_i$, $\theta_i^{UB}$ (the upper bound on $\theta_i$), $\tilde{\epsilon}_{im}$
Output: Set $S$ of $(\tau_i, X_m^i, \theta_i, \eta(X_m^i, \theta_i))$
for $\tau_i \in \mathcal{T}$ do
  $\theta_i = 0$
  while $\theta_i < \theta_i^{UB}$ do
    decide a set of $\epsilon_{im}$ that satisfies $\mathcal{MC}_i$
    compute $G_i(\mathcal{MC}_i, \theta_i)$
    for $X_m^i \in \mathcal{MC}_i$ do
      if $X_m^i$ is a BD state in $\mathcal{MC}_i$ then
        $\epsilon_{im}^* = (1 - \tilde{\epsilon}_{im})$;
      end if
      if $X_m^i$ is an MF state in $\mathcal{MC}_i$ then
        $\epsilon_{im}^* = \tilde{\epsilon}_{im}$;
      end if
      if $X_m^i$ is an BF state in $\mathcal{MC}_i$ then
        $\epsilon_{im}^* = \tilde{\epsilon}_{im} + \epsilon_{im}$;
      end if
      compute $\theta_i^*$;
      evaluate $G_i(\mathcal{DP}_i^*, \theta_i^*)$;
      compute $\eta$ according to Equation (2);
      put $(\tau_i, X_m^i, \theta_i, \eta)$ in $S$;
    end for
    $\theta_i += \Delta$;
  end while
end for
```

Algorithm 1 follows naturally from the discussions in Section 3.2. The first **if** statement checks if state $X_m^i$ is a BD state, e.g., state $\{10\}$ in Figure 1(c). If it is, an extra transition is introduced from $X_m^i$ to a state such that the next job is not dropped, e.g., state $\{01\}$ in Figure 1(c). The next **if** statement is similar to this except that it deals with MF states. We introduced $\Delta$ as the quantizer to discretize $\theta_i$. The choice of $\Delta$ value impacts the size of the set of sensitivity values. Our experiments have shown that $\Delta$=0.01 is a reasonable choice. The choice of $\tilde{\epsilon}_{im}$ should reflect how the current dropout process deviates from $\mathcal{MC}_i$. In our experiment we have tried different $\tilde{\epsilon}_{im}$ values from 0.1 to 0.3. The sensitivity values remain similar. Due to page limit, we only show the scheduling results of $\tilde{\epsilon}_{im} = 0.1$.

Algorithm 2 first assigns a newly released job to one of the three groups according to the job's current state, $X_m^i$. Within each group, the priority of a job is determined by looking up the table containing the sensitivity values. Algorithm 2 is called at every job release, completion and drop. At a job completion or drop, the work is minimal. At a job release, the time and space complexity of algorithm 2 are $O(N \log(N))$ and $O(NMR)$ respectively, where $N$ is the number of tasks, $M$ is the maximum number of states among all given MC constraints, and $R$ is the maximum number of discrete average-dropout rate values. Usually,

**Algorithm 2** SSA-online

```
Input: Set $S$ of $(\tau_i, X_m^i, \theta_i, \eta) \forall \tau_i \in \mathcal{T}$
At a scheduling point $t$
for each job $\tau_{ij}$ either completed or dropped at $t$ do
  update $X_m^i$ /* the state of $\tau_i$'s dropout process */
  update $\theta_i$;
end for
for each job $\tau_{ij}$ released at $t$ do
  if $X_m^i$ is an MF (resp., BF, BD) state then
    if there are other jobs in MF (resp., BF, BD) group then
      look up $\eta(X_m^i, \theta_i)$ in $S$;
      if $\tau_{ij}$ is in MF (resp., BF) state then
        insert $\tau_{ij}$ in MF (resp., BF) group in decreasing order of $\eta$;
      else
        insert $\tau_{ij}$ in BD group in *increasing* order of $\eta$;
      end if
    else
      put $\tau_{ij}$ into MF (resp., BF, BD) group;
    end if
  else
    if dropping $\tau_{ij}$ makes $X_m^i$ a state in $\mathcal{MC}_i$ then
      discard $\tau_{ij}$;
    else
      put $\tau_{ij}$ at the head of MF group;
    end if
  end if
end for
```

the number of states in MC constraints is not large (less than 10). The value $R$ reflects a trade off between scheduling overhead and scheduling quality and is application dependent. Our experiments based on implementing the algorithm in the QNX environment indicates that the overhead is not significant for the control tasks considered. If the number of tasks is large, clever storage of the sensitivity table could help reduce the overhead, which is beyond the scope of this paper.

## 3.4 Grouped Fair Dropout Rate algorithm

SSA prioritizes jobs inside each priority group according to jobs' state sensitivities, which reflect jobs' criticalities. In scheduling tasks with MC constraints, the "fairness" of allocating resources should also be taken into consideration to achieve better overall system performance. However such "fairness" should only be achieved under the condition that each task is following their MC constraints. To satisfy such "fairness" and the MC constraints at the same time, we introduce GFDR. GFDR is a purely online algorithm which partitions jobs into MF, BF, BD groups like SSA does. It also monitors the average dropout rates $\theta_i$ of each task. Instead of using the state sensitivity, GFDR uses task's "weighted" average dropout rate $(w_i \times \theta_i)$ to prioritize jobs inside each group. The larger the "weighted" average dropout rate, the higher the priority. Due to the page limit, we omit the details of GFDR. The time complexity of GFDR is the same as that of SSA. The space complexity of GFDR is $O(N)$, which is much less than that of SSA.

## 4 Experimental Results

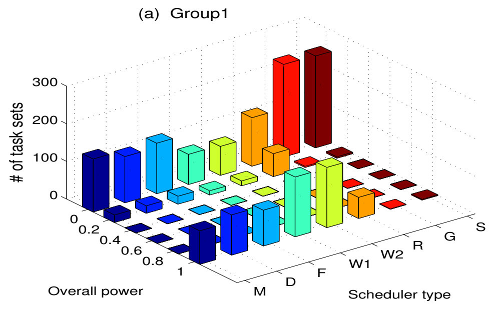
In this section, we first present our simulation setup and the results obtained for randomly generated tasks. To further assess the applicability of our scheduling approach, we have also implemented our algorithms in a real-time OS. The experimental setup and results are shown next.

### 4.1 Simulation

Our goal of simulation is to evaluate the effectiveness of SSA and GFDR. A discrete-event simulator was developed to simulate job execution as well as the online scheduler.

In the experiments, we randomly generated a large number of task sets, each of which contains 5, 10 or 20 tasks. The period of each task was randomly selected from a uniform distribution between 2 to 50 time unit, and the deadline of each task was set to be the same as its period. The execution time of every task was also randomly generated. In each task set, some of the tasks are associated with the MC constraint given in Figure 1(c) (referred to as MC-1), which is the optimal dropout process for the NCS model in Figure 2(top). Other tasks are associated with the MC constraint in Figure 1(b) (referred to MC-2), which is the optimal dropout process of an "inverted pendulum" control system [14]. The performance of a task is measured by the output signal power and lower signal power means better performance. We assume that each task is equally important to the overall system performance when the output signal of each task is below some upper bound. If the output signal power of any task in the task set is above the bound, the whole system is considered unacceptable. This assumption reflects the importance of coordination among the tasks. For example, in an airplane, every individual control task must be stable. If any of the individual control task became unstable, the airplane would malfunction and might even crash. The upper bound in our experiment is set to the stable boundary of the two underlying control systems. For each task in a task set, we recorded the completion status of 1 million jobs and used `Matlab Simulink` to compute the underlying control task performance based on the completion status. Then we applied equation ( 1) to determine the overall system performance.

To evaluate the effectiveness of SSA and GFDR, it is necessary to identify algorithms against which SSA and GFDR should be compared. The algorithms, MDA, DDA and FDA, in [16] are natural choices. For completeness, we also compare our algorithms with other two algorithms, WHSA and FDR, which are possible alternatives. WHSA (weakly hard system scheduling algorithm) [7] is one of the most powerful and efficient scheduling algorithms for tasks with windowed based constraint. It partitions jobs into two priority groups online to ensure certain dropout patterns. FDR (fair average dropout rate) scheduling algorithm uses



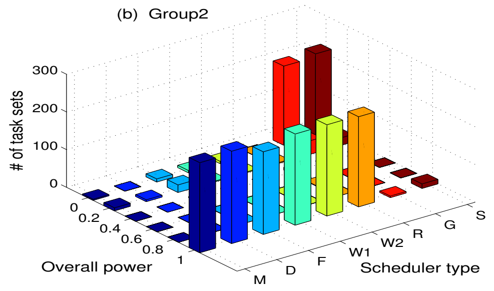


**Figure 4.** Histograms illustrating overall output signal power for task sets with utilization between 1.0-1.2 (group1) and 1.2-1.4 (group2). On the "scheduler type" axis, "M", "D", "F", "w1", "w2", "R", "G", "S" represent MDA, DDA, FDA, WHSA1, WHSA2, FDR, GFDR and SSA respectively. Each "slice" of the "scheduler type" axis is a histogram of task sets' overall output signal powers resulted from using that type of scheduler.

the current average dropout rate as task priority, i.e., a task with higher average dropout rate gets a higher priority. FDR aims at ensuring fair usage of resource by tasks in terms of average dropout rates when the system is overloaded. To compare with WHSA, we need to represent the MC constraints by window-based constraints for WHSA to use. The rationale in choosing the constraints is that they should be as "close" to the corresponding MC constraints as possible. For constraint MC-1, we may use either $\langle 2, 4\rangle$ (completing two consecutive jobs in any 4-job window) or (2,4) (completing any two jobs in any 4-job window). For MC-2, (2,3) (completing any two jobs in any 3-job window) is the most appropriate window based constraint. In the experimental results, WHSA1 employs $\langle 2, 4\rangle$ for MC-1 and (2,3) for MC-2, and WHSA2 uses (2,4) for MC-1 and (2,3) for MC-2.

Figure 4(a) and (b) plot the distribution of the overall output signal power (weighted sum of the normalized output signal power of each task) of two groups of task sets for each scheduling algorithm. Each group contains 250 task sets, and each task set consists of 5 tasks. In group1, each task set's utilization (or the load it presents to the system) is between 1.0 and 1.2. And in group2 it is between 1.2 and

1.4. Experimental results for task sets with other numbers of tasks and other utilization are omitted due to space limit. From Fig. 4, one can observe that the overall output signal power values of task sets scheduled by GFDR and SSA tend to be lower, while those of other algorithms tend to be larger. (Recall that for the control tasks in the experiments, lower output signal power indicates better performance.) If the overall output signal power of a task set equals 1, it indicates that the output signal power of some task is beyond the upper bound and the system is considered to be unstable (unacceptable). In both groups of task sets, GFDR and SSA, unlike other schedulers, rarely result in unstable systems. The better performance of GFDR and SSA can be attributed to the fact that both of them employs the 3-way partitioning scheme to enforce the MC constraints. Fig. 4 also show that as the task set's utilization (system load) increases, GFDR and SSA still manage to result in good performance while the performance of other schedulers degrades much faster. In fact, when the system load is larger than 1.2, the performances due to other schedulers are rarely acceptable (less than 1% of total number of task sets).

To examine the performance difference due to GFDR and SSA, we plot the stable task set percentage and the average overall output power at heavier system loads (task set's utilization is larger than 1.4) in figure 5(a) and (b). From figure 5, one can see that SSA indeed is better than GFDR. Figure 5(a) shows that as the system load increases, the stable task set percentage of GFDR decreases much faster than that of SSA. For example, when the utilization is between 1.6 and 1.7, SSA doubles the stable task set percentage achieved by GFDR. In figure 5(b), one can see that the average overall output signal power due to SSA is also better than that of GFDR. For the group of task sets whose utilization is between 1.6 and 1.7, the improvement is up to 33%. The better performance of SSA can be attributed to the careful offline analysis which leads to a more effective scheduling decision online. Note that state sensitivity is a function of average dropout rate $\theta$. Therefore, SSA also takes the "fairness" into consideration.

## 4.2 Implementation in a real-time OS

Though the simulation results indicate the superiority of SSA and GFDR, it is important to assess the applicability of SSA and GFDR in practical situations. Towards this goal, we have implemented all the scheduling algorithms discussed in the previous subsection in a real-time operating system, QNX4.25 [13].

Figure 6 gives an overall view of our experimental setup. The SSA-online, GFDR and other priority assignment schemes are implemented as a process whose priority is lower than QNX system tasks but higher than application tasks. It is invoked whenever a new job is released. The application models a networked control system consisting of five control tasks which run at the application level. Each control task performs the necessary computation for the plant to be controlled. In addition, it simulates the delay, assumed to be a random number, incurred by the network. Instead of directly connecting the computer with physical control plants (such as motors), we have used five processes to model the five plants. Use of computer models for physical plants allows us to monitors plant behavior in detail. Moreover, this step is necessary for validating control algorithms and avoiding any potential damage to physical plants. Each control plant model is implemented as a process that simulates the runtime behavior of the respective physical plant. Three of the control plant processes model the control system in Figure 2(top), while the rest two model "inverted pendulum" control system [14]. Each control-plant process runs at a certain frequency dictated by the physical plant and has a priority level equal to the system tasks in order to ensure adequate monitoring of the control-plant behavior.

We implemented all the scheduling algorithms discussed

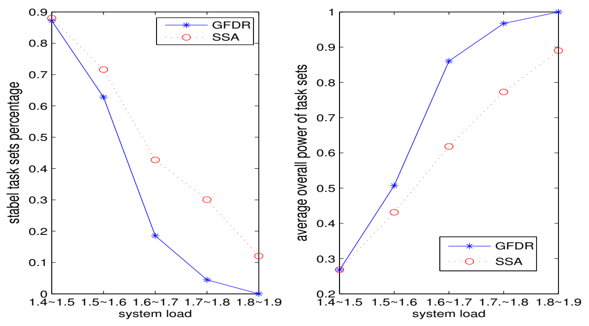


**Figure 5.** GFDR vs SSA.

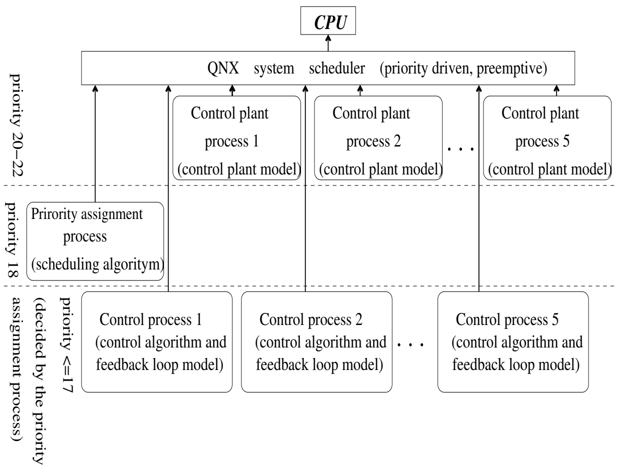

CPU
QNX system scheduler (priority driven, preemptive)
Control plant process 1 (control plant model)
Control plant process 2 (control plant model)
Control plant process 5 (control plant model)
Prirority assignment process (scheduling algoritym)
Control process 1 (control algorithm and feedback loop model)
Control process 2 (control algorithm and feedback loop model)
Control process 5 (control algorithm and feedback loop model)
priority 20~22
priority 18
priority <=17 (decided by the priority assignment process)

**Figure 6.** Experimental setup based on QNX.

in Section 4.1 in the above setup. For each algorithm, we ran more than 20 experiments (for different network delay values). Each experiment took several hours so that each control plant model could release at least 100,000 jobs (the period of each control plant is between 20 ms and 500 ms). The "output signal power" of each control-plant process was then collected and used to compute the overall system performance. All experiments were carried out on a 266 MHz Pentium II. Due to space limit, we omit these simulation results. The experimental results again indicate that GFDR and SSA outperform other priority assignment schemes. We have also simulated the systems in this study with our discrete-event simulator and the results obtained agree well with the data collected here. The experiments confirm that our scheduling approaches are indeed applicable for realistic control tasks and their overhead can be readily tolerated.

## 5 Conclusion and future work

In this paper, we address the scheduling problem of a FRTS containing tasks associated with MC constraints. We present two heuristic scheduling approaches that exploit the unique features of MC constraints. SSA combines offline analysis with online priority assignment to ensure that overloaded systems degrade gracefully without excessive scheduling overhead. The introduction of the state sensitivity concept allows us to quantify the deviation of a stochastic process from another one and thus derive an effective priority assignment scheme. GFDR is a simple heuristic algorithm to balance tasks' performance under MC constraints. Simulation results show that both our approaches outperform others significantly and that SSA outperforms GFDR under heavy system load. Implementation of our approaches in a real-time operating system environment further validates the work. As future work, we intend to study approaches based on admission control to handle MC constraints. We would also like to derive performance guarantees for systems with MC constraints.

## References

[1] T.F. Abdelzaher, K. Shin and N. Bhatti "Performance guarantees for Web server end-systems: a control-theoretical approach," *IEEE Trans. on Par. & Distr. Sys.*, Vol.13, No.1, 2002, pp. 80-96.

[2] A. Atlas and A. Bestavros, "Statistical rate monotonic scheduling," *Real-Time Systems Symposium*, 1998.

[3] A. Atlas and A. Bestavros, "Slack stealing job admission control scheduling," Technical Report BUCS-TR-1998-009, Boston University, Computer Science Dept., 1998.

[4] S. Baruah, G. Korean, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha and F. Wang, "On the competitiveness of on-line real-time task scheduling," *Real-Time Systems*, Vol. 4, 1992, pp. 125-144.

[5] S. Baruah, J. Haritsa and N. Sharma, "On-line scheduling to maximize task completions," *Real-Time Systems Symposium*, 1994, pp. 228-237.

[6] G. Bernat, A. Burns and A. Llamosi "Weakly hard real-time systems," *IEEE Trans. on Computers*, Vol. 50, No. 4, 2001, pp. 308-321.

[7] G. Bernat, R. Cayssials "Guaranteed On-Line Weakly-Hard Real-Time Systems," *Proceedings of Real-Time Systems Symposium*, 2001,pp. 25-34

[8] G. Bernat and A. Burns, "Combining (n,m)-hard deadlines and dual priority scheduling," *Real-Time Systems Symposium*, 1997, pp. 46-57.

[9] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k)-firm deadlines," *IEEE Trans. on Computers*, 1995, 44, pp. 1443-1451.

[10] G. Koren and D. Shasha, "Skip-over: Algorithms and complexity for overloaded systems that allow skips," *Real-Time Systems Symposium*, 1995, pp. 110-117.

[11] Aloysius K.Mok, Weirong Wang, "Window-contrainted Real-Time Periodic task Scheduling", *Proceedings of Real-Time Systems Symposium*, 15-24, 2001.

[12] C. Lee, J. Lehoczky, R. Rajkumar and D. Siewiorek, "On quality of service optimization with discrete QoS options," *Real-time Technology and Applications Symposium*, 1999.

[13] http://www.qnx.com.

[14] http://www.engin.umich.edu/group/ctm/examples /pend/invpen.html.

[15] Q. Ling and M.D. Lemmon, "Soft real-time scheduling of networked control systems with dropouts governed by a Markov chain," *American Control Conference*, June 2003.

[16] D. Liu, X. Hu, M.D. Lemmon, Q. Ling "Firm real-time system scheduling based on a Novel QoS constraint," *Real-Time Systems Symposium*, 2003,pp. 386-396

[17] C. Lu, j.A. Stankovic, S.H. Son and T. Gang, "Feedback control real-time scheduling: framework, modeling, and algorithms," *Real-Time Systems*, Vol.23, No.1-2, July-Sept. 2002, pp.85-126.

[18] R. Rajkumar, C. Lee, J. Lehoczky and D. Siewiorek, "A resource allocation model for QoS management," *Real-Time Systems Symposium*, 1997.

[19] R. Rajkumar, C. Lee, J. Lehoczky and D. Siewiorek, "Practical solutions for QoS-based resource allocation problems," *Real-Time Systems Symposium*, 1998.

[20] P. Ramanathan, "Overload management in real-time control applications using (m,k)-firm guarantee," *IEEE Trans. on Par. & Distr. Sys.*, 1999, 10(6), pp. 549-559.

[21] T.-S. Tia, *et. al.* "Probabilistic performance guarantee for real-time tasks with varying computation times", *Real-Time Technology and Applications Symposium*, 1995, pp. 164-173.

[22] R. West and K. Schwan, "Dynamic window-constrained scheduling for multimedia applications," *International Conference on Multimedia Computing and Systems*, 1999.