Multiplexing VBR Traffic Flows with Guaranteed Application-level QoS Using Statistical Rate Monotonic Scheduling*

Alia Atlas[†]
Dept of Internetwork Research
BBN Technologies
Cambridge, MA 02138

Azer Bestavros
Computer Science Department
Boston University
Boston, MA 02215

{akatlas, best}@cs.bu.edu

Abstract

The data units transmitted by an application may vary in size while being constant in rate, which results in a variable bit rate (VBR) data flow. That data flow requires QoS guarantees. Statistical multiplexing is inadequate, because no guarantees can be made and no firewall *property exists between different data flows. In this paper, we present a novel resource management paradigm for the maintenance of application-level QoS for VBR flows. Our paradigm is based on Statistical Rate Monotonic Scheduling (SRMS), in which (1) each application generates its variable-size data units at a fixed rate, (2) the partial delivery of data units is of no value to the application, and (3) the QoS guarantee extended to the application is the probability that an arbitrary data unit will be successfully transmitted through the network to/from the application.*

1. Introduction

There are many applications, such as video, with periodic *message* transmissions where (1) the *message* sizes are variable, (2) the **entire** message must be received for the transmission to be useful, and (3) not all messages must be received to support acceptable functionality. If such an application were to reserve its peak rate, the network would have very poor utilization and would refuse many other reservation requests. Instead, a VBR reservation with a QoS guarantee is preferable.

In [4] we have introduced Statistical Rate Monotonic Scheduling (SRMS)—an algorithm that allows for the efficient scheduling of periodic real-time task systems with statistical QoS guarantees. In this paper, we present an SRMS-based paradigm for multiplexing many VBR data flows across a constant bandwidth link while supporting QoS for each data flow. SRMS lends itself very well to communication systems due to its ability to cope with variable (rather than deterministic) resource consumption requirements, its ability to manage tasks with QoS guaranteed best-effort deadlines, and its support of the *firewall* property. Our paradigm incorporates a number of unique features and novel capabilities, including: (1) fixed priority scheduling that takes into account both task criticality and periodicity, (2) message admission control that allows for early rejections of messages that are not guaranteed to meet their specified QoS, thus preserving resources, (3) integration of reservation-based and best-effort resource scheduling seamlessly, and (4) controllable graceful degradation under conditions of overload.

The problem of scheduling multiple data streams across a single link has been extensively studied. A comprehensive literature review of such studies is given in [4].

2. Network Model and SRMS Framework

Our network model in this research consists of border switches which are connected to each other via an arbitrary network. Each border switch handles a large number of data flows from an internal network. A CBR connection, or a virtual circuit, is assumed to exist between any two border switches which must communicate. The situation described is depicted in Figure 1. Each application is assumed to generate application-level data units, known as *messages*, at a constant rate, R_i. The messages are of variable size. The message flow can be modeled as a periodic task with a variable resource requirement. The period of the message flow is $\frac{1}{R_i}$. At the beginning of each period, a complete message is ready to be sent.

We assume that the number of applications generating traffic which need to be routed through a border switch is significantly greater than the number of CBR connections and virtual circuits which are established from that border switch. Therefore, many different message flows will need to be switched to the same output link. It is necessary to schedule which cells are selected to be transmitted. Therefore, at each output link, buffering and scheduling are necessary. A buffer which can hold two maximum-length messages is required for each message flow. The buffer will hold the incoming message, to be sent out the next period, and the outgoing message. To conserve buffer space and to minimize delay, each message must be fully transmitted by the end of the period at which it was ready to be sent.

With this deadline restriction, the traffic flow representing messages generated from an application resembles a classical real-time periodic task model, with two differ-

*This work was supported by NSF research grant CCR-9706685.
[†]Research completed while co-author was at Boston University.

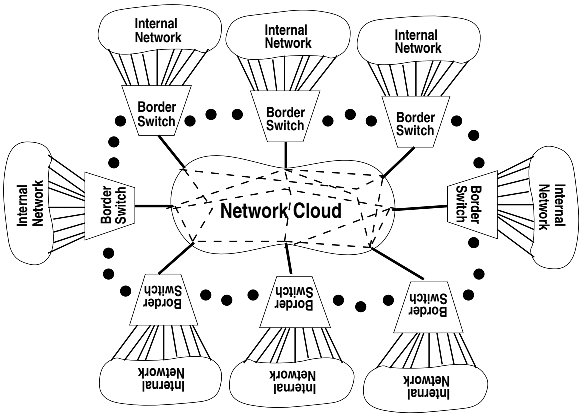

Figure 1. Model of Network

ences. First, the resource requirement is variable. Second, if a message cannot be sent by its deadline, then the entire message should be dropped; this is known as a *firm* deadline. The additional requirement of the application is that some QoS guarantee be provided.

In the remainder of this paper, we will consider each message flow to represent a periodic task. Namely, a periodic task system represents a set of VBR traffic flows that are multiplexed on a single CBR channel. Thus, throughout the paper, we use the terms *"task" ("resource")* and *"VBR traffic flow" ("CBR channel")* interchangeably.

2.1. Statistical Rate Monotonic Scheduling

A periodic task, τ_i, is a three-tuple, $(P_i,\ f_i(x),\ QoS_i)$, where P_i is the task's period, $f_i(x)$ is the probability density function (PDF) for the resource requirement (message size), and QoS_i is the task's guaranteed QoS. The quality of service for a task (message flow) τ_i is defined as the probability that an arbitrary job (message) of τ_i will be completed (transmitted) by its deadline. We denote this probability by QoS_i.

RMS and SRMS are both preemptive scheduling algorithms. A cell cannot be preempted while it is being transmitted. Therefore, all periods must be multiples of CT, the amount of time it requires to transmit a single cell, given the bandwidth of the outlink. This requirement can be reinforced by setting $P_i = \lfloor \frac{\frac{1}{R_i}}{CT} \rfloor$. Therefore, all preemptions occur at the end of a cell's transmission.

Without loss of generality, we assume that tasks are ordered rate monotonically. Task 1, τ_1, is the task with the shortest period, P_1. The task with the longest period is τ_n, where n is the total number of tasks in the system. The shorter the period, the higher the task's priority. At the start of every P_i units of time, a new message of task τ_i (a *job* of task τ_i) is available and has a firm deadline at the end of that period. Thus, the jth job of task i—denoted by $\tau_{i,j}$—is released and ready at time $(j-1)*P$ and its firm deadline is at time $j*P$.

We assume that the resource requirements for all jobs of a given task are independent and identically distributed (iid) random variables. The distribution is characterized using the probability density function (PDF), $f(x)$. We assume that the resource requirement for a job (message size) is known when the job is released and that such a requirement is accurate.The resource requirement for the jth job of the ith task is denoted by $e_{i,j}$.

The third element of a task specification under the SRMS paradigm is its QoS requirement. Using the methods presented in this paper, this QoS requirement can be used to determine the necessary allowance needed to guarantee the QoS. The allowance a_i is the amount of time alloted to task, τ_i, over an epoch of time equal to the period of the next lower priority task τ_{i+1}. If the allowance a_i is specified instead of QoS_i, then the QoS of the task with that allowance is $QoS(\tau_i)$.

The superperiod of τ_i is P_{i+1}, the period of the next lower priority task, τ_{i+1}. A job $\tau_{i,j}$ whose release time is in one superperiod and whose deadline is in the next superperiod is called an *overlap* job. The utilization requirements of overlap jobs could be satisfied through the use of allowances disbursed within either (or both) superperiods, whereas the utilization requirements of a non-overlap job must be satisfied through the use of the allowance disbursed within a single superperiod—namely the enclosing superperiod.

A set of tasks $\tau_1, \tau_2, ..., \tau_n$ is said to be schedulable under SRMS, if every task τ_i is guaranteed to receive its allowance a_i at the beginning of every one of its superperiods.

In SRMS, the highest priority admitted job is scheduled. SRMS maintains a *budget* for each task in the system. Jobs belonging to a task are allowed to use the resource, if there is enough budget for them to do so. More specifically, at the beginning of the superperiod of task τ_i, the *budget* of τ_i is replenished to τ_i's allowance (namely a_i).

Upon the release of a non-overlap job $\tau_{i,j}$, if the resource requirement of that job, namely $e_{i,j}$, is less than the remaining budget for the current superperiod, then job $\tau_{i,j}$ is admitted and the remaining budget for the current superperiod is decreased by an amount equal to $e_{i,j}$. If $e_{i,j}$ is more than the remaining budget for the current superperiod, then job $\tau_{i,j}$ is *not* admitted and the remaining budget for the current superperiod remains unchanged. However, if job $\tau_{i,j}$ is an overlap job, then it may still be possible to admit that job by delaying its service—assuming that such a delay does not result in missing $\tau_{i,j}$'s deadline—until the start of the next superperiod, at which time the budget is replenished, and admission may be possible.

There are many issues that we have not discussed with regard to SRMS, including specific optimizations. For more details, interested readers should refer to our presentation of SRMS and its extensions in [4].

3. QoS Management using SRMS

With the brief description of SRMS presented in the previous section, we are now ready to discuss our SRMS-based QoS management paradigm. First, we will consider how to calculate the QoS of a task, given a set allowance. For simplicity, we shall discuss calculating the QoS for task systems with harmonic periods and provide a trivial example. In subsection 3.2, we present a generalization for task systems with arbitrary periods. Finally, we will discuss calculating the allowance from QoS requirement.

3.1. QoS for Harmonic Task Systems

A task set is harmonic if, for any two tasks τ_i and τ_j, $P_i < P_j \Rightarrow P_i | P_j$. Under SRMS, a necessary and sufficient condition [4] for a harmonic task set to be schedulable is that $\sum_{\forall i} \frac{a_i}{P_{i+1}} \leq 1$.

Lemma 1 *Given a schedulable, harmonic task set $\tau_1, \tau_2, ..., \tau_n$, the maximum possible resource utilization requirement for any job of task τ_i is e_i^{maxp}, where:*

$$e_i^{maxp} \leq P_i - \sum_{j=1}^{i-1} \frac{a_j * P_i}{P_{j+1}}$$

The proof of this lemma follows from the fact that the task set is schedulable, and hence every task τ_j that has a priority higher than that of τ_i—namely $\tau_j, j < i$—must be able to claim its allowance for every superperiod of τ_j that occurs within a single period of task τ_i.

Because SRMS uses job admission control, a task is not affected by the variability in the resource utilization of the other tasks in the system. Therefore, each task can be given seperate statistical guarantees.

As illustrated in figure 2, a job $\tau_{i,j}$ can fall into $\frac{P_{i+1}}{P_i}$ different *phases* within the superperiod P_{i+1}. The probability that $\tau_{i,j}$ will be admitted is dependent on the phase in which it falls. To explain this, it suffices to observe that the first job in the superperiod has a replenished budget and has the best chance of making its deadline, while the last job in the superperiod has a smaller chance, because the budget is likely to have been depleted.

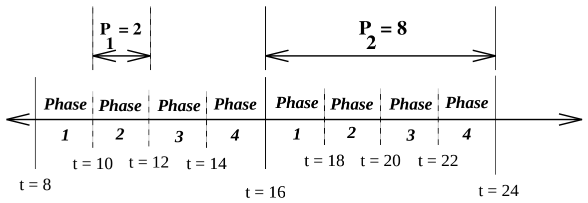

Figure 2. Sample Task with Four Phases

An arbitrary job $\tau_{i,j}$ has an equal probability of being in any given phase out of the possible $\frac{P_{i+1}}{P_i}$ phases within the superperiod P_{i+1}. To explain this, it suffices to note that in an infinite execution of task τ_i, there will be an equal number of jobs in each phase, and thus a uniform distribution for the phase of a randomly selected job is reasonable.

Let $S_{i,k} = 1$ ($S_{i,k} = 0$) denote the event that a job $\tau_{i,j}$ released at the beginning of phase k of a superperiod of task τ_i is admitted (not admitted) to the system. Now, we proceed to compute $P(S_{i,k} = 1)$—the probability of admitting a job in the kth phase of a superperiod of task τ_i (i.e. the probability of success).

Recall that a_i is the allowance made available to task τ_i at the start of its superperiod P_{i+1}, which is the start of the first phase. Obviously, a job $\tau_{i,j}$ released in this first phase (i.e. $k = 1$) will be admitted only if its requested utilization is less than or equal to a_i. This leads to the following relationship.

$$P(S_{i,1} = 1) = P(e_{i,j} \leq a_i)$$

For a job $\tau_{i,j}$ released in the second phase (i.e. $k = 2$), two possibilities exist, depending on whether the job released in the first phase was admitted or not admitted. This leads to the following relationship.

$$\begin{aligned} P(S_{i,2} = 1) = P(e_{i,j-1} \leq a_i) * P(e_{i,j-1} + e_{i,j} \leq a_i) \\ + P(e_{i,j-1} > a_i) * P(e_{i,j} \leq a_i) \end{aligned}$$

Obviously, each $P(S_{i,k} = 1)$ can be calculated as the sum of 2^{k-1} different terms, where each term expresses a particular history of previous jobs being admitted and/or rejected (i.e. deadlines met and/or missed). Thus, to calculate $P(S_{i,3} = 1)$, the sum of the probabilities of all possible histories, where the job in the third phase meets its deadline, must be calculated. The set of possible histories are ((1,1,1), (1,0,1), (0,1,1), (0,0,1)), where 1 represents a met deadline and 0 represents a missed deadline.

We are now ready to define the QoS guarantee that SRMS is able to extend to an arbitrary set of tasks with harmonic periods.

Theorem 1 *Given a task set with harmonic periods, the probability than an arbitrary job $\tau_{i,j}$ of task τ_i will be admitted is the QoS function of τ_i.*

$$QoS(\tau_i) = \frac{P_i}{P_{i+1}} * \sum_{k=1}^{\frac{P_{i+1}}{P_i}} P(S_{i,k} = 1)$$

Theorem 1 follows from the assumption that an arbitrary job has an equal probability of being in any given phase. The value thus calculated, $QoS(\tau_i)$, is the statistical guarantee which harmonic SRMS provides on the probability that an arbitrary job will miss its deadline.

To illustrate the use of the above formulas, consider the example task system shown in Table 1. The results of applying Theorem 1 to that system are shown in Table 2.

i	P_i	E_i^{max}	$E(E_i)$	PDF	# Phases
1	5	2	1.5	uniform	2
2	10	3	2	uniform	3
3	30	13	7	uniform	∞
4	90	4	2.5	uniform	1

Table 1. Example: 4 Tasks, Max Utilization 1.178

What do these calculations mean? Because the periods are harmonic, all of the processor time can be guaranteed. Therefore, the allowances a_1, a_2, a_3 and a_4 could be set to any set of values, as long as the total utilization is not greater than 1 (i.e. $\sum_{i=1}^{4} \frac{a_i}{P_{i+1}} \leq 1$. Table 3 shows a number of feasible resource assignments and the associated QoS delivered to the various tasks in the system. Obviously, the choice of a particular assignment should reflect the importance of the different tasks.

3.2. QoS for Arbitrary Task Systems

The calculation of QoS for a task system with arbitrary periods is an elaboration of the QoS calculation for a harmonic task system. The additional complexity is caused by an analysis of the behavior for overlap jobs. Recall that, according to SRMS, when an overlap job, $\tau_{i,j}$, is released, that job may be delayed for a bounded time. After that

Guarantee Calculations for Task 1

a_1	$P(S_{1,1}=1)$	$P(S_{1,2}=1)$	$QoS(\tau_1)$
2	1	$\frac{1}{4}$	$\frac{5}{8}$
4	1	1	1

Guarantee Calculations for Task 2

a_2	$P(S_{2,1}=1)$	$P(S_{2,2}=1)$	$P(S_{2,3}=1)$	$QoS(\tau_2)$
3	1	$\frac{1}{3}$	$\frac{19}{81}$	0.523
6	1	1	0.6296	0.877
9	1	1	1	1

Guarantee Calculations for Task 3

a_3	$P(S_{3,1}=1)$	$P(S_{3,2}=1)$	$P(S_{3,3}=1)$	$QoS(\tau_3)$
21	1	0.911	0.5628	0.825
24	1	0.982	0.701	0.8944
27	1	1	0.834	0.9448
30	1	1	0.925	0.975
33	1	1	0.9745	0.9915
36	1	1	0.995	0.998
39	1	1	1	1

Table 2. Example: QoS Calculations

a_1	a_2	a_3	a_4	Util	QoS_1	QoS_2	QoS_3	QoS_4
4	9	24	3	1	1	1	0.8944	0.75
4	3	39	4	0.9778	1	0.523	1	1
2	9	39	4	1	0.625	1	1	1
4	6	33	3	1	1	0.877	0.9915	0.75

Table 3. Example: Valid Resource Assignments

delay, the task budget is renewed and the overlap job is tested for admittance.

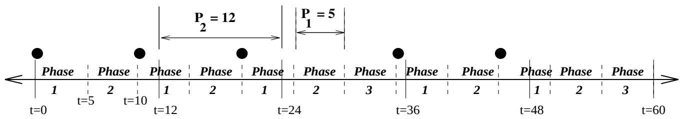

Figure 3. Phases for Task with Overlap Jobs

In Figure 3, an example task is shown with the various phases. Each black circle represents a renewal of the budget and, potentially, a delayed job release time. For more details on the algorithm, please see [4]. As can also be seen in Figure 3, the largest possible number of jobs which might need to share an allowance is $\lceil \frac{P_{i+1}}{P_i} \rceil$. This worst-case occurs, for example, when the deadline of the last job in the superperiod is also the end of the superperiod. The smallest possible number of jobs is $\lfloor \frac{P_{i+1}}{P_i} \rfloor$. This occurs when the release time of a job corresponds to the start of a superperiod. Exactly how many times each of these cases occur can be calculated as follows.

$$highCount + lowCount = \frac{LCM(P_{i+1}, P_i)}{P_{i+1}}$$

$$highCount * \lceil \frac{P_{i+1}}{P_i} \rceil + lowCount * \lfloor \frac{P_{i+1}}{P_i} \rfloor = \frac{LCM(P_{i+1}, P_i)}{P_i}$$

where, $LCM(P_{i+1}, P_i)$ is the least common multiple of the two periods. HighCount is the number of superperiods in $LCM(P_{i+1}, P_i)$ in which there are $\lceil \frac{P_{i+1}}{P_i} \rceil$ phases. Similarly, lowCount is the number of superperiods in $LCM(P_{i+1}, P_i)$ in which there are $\lfloor \frac{P_{i+1}}{P_i} \rfloor$ phases.

The first equation expresses the fact that highCount plus lowCount must equal the total number of superperiods in the $LCM(P_{i+1}, P_i)$. The second equation describes the number of jobs in the $LCM(P_{i+1}, P_i)$. Both highCount and lowCount are weighted by the number of jobs they represent. The weighted sum must equal the total number of jobs in $LCM(P_{i+1}, P_i)$. By solving these two equations, highCount and lowCount can be determined.

Job admission for a task set with arbitrary periods proceeds through two tests. The first test is a check that the sum of allocated execution times during the superperiod is less than or equal to the task's allowance. Thus, the probability that a job will be able to meet its deadline (i.e. $P(S_{i,k} = 1)$) is equal to the sums of the probabilities of the possible histories. The second test for job admission exists because an overlap job (that passed the first test) may have been delayed so long that it is impossible to meet its deadline (i.e. even if admitted). Therefore, in the probability calculations for each possible history, the value $P(e_{i,1} \le a_i)$ ($P(e_{i,1} > a_i)$) used in the harmonic case is conditioned by the probability that the second admission test is passed. Given this slight complication, the probability that a job in the jth phase of τ_i is admitted, $P(S_{i,k} = 1)$, is still the sum of the probabilities of all possible histories, where a job in the kth phase meets its deadline.

To calculate the probability that a job in the first phase will pass the second admission test, we make several assumptions. First, we assume that the resource requirement of the job associated with the superperiod, $e_{i+1,m}$, is the maximum schedulable. This assumption also ensures that no cascading priority inversion can occur [6]. Second, we assume that the deadline for that superperiod job corresponds to the deadline of a job of τ_i. This is the worst case, because it requires all of the allowance allotted for the superperiod to be spent during the actual superperiod and the number of phases in that superperiod to be $\lceil \frac{P_{i+1}}{P_i} \rceil$.

The maximum schedulable resource requirement for τ_i is slightly different from the case with harmonic periods and is given by:

$$e_i^{maxp} = P_i - \sum_{k=1}^{i-1} a_k * \lceil \frac{P_i}{P_{k+1}} \rceil$$

The remaining resource requirement of a job $\tau_{i,j}$ at a time t is represented by $e_{i,j}^{left}(t)$.

The probability that the first task is admitted is $P(e_{i+1,k}^{left}((j-1) * P_i) + e_{i,j} \le e_i^{maxp})$. The PDF for $e_{i,j}$ is known. Next, we need to determine the PDF of $e_{i+1,k}^{left}((j-1) * P_i)$, the remaining resource requirement for job $\tau_{i+1,k}$ when the overlap job, $\tau_{i,j}$ is released. The worst case resource requirement for a job of τ_{i+1} is e_{i+1}^{maxp}.

At $(j-1) * P_i$, the time when the overlap jth job is released, the minimum resource requirement spent on the kth job of task τ_{i+1} is given by:

$$e_{i+1,k}^{spent} = (j-1) * P_i - (k-1) * P_{i+1} - a_i - \sum_{m=1}^{i-1} a_m * \lceil \frac{(j-1) * P_i - (k-1) * P_{i+1}}{P_{m+1}} \rceil$$

$$= P_i * \lfloor \frac{P_{i+1}}{P_i} \rfloor - a_i - \sum_{m=1}^{i-1} a_m * \lceil \frac{P_i * \lfloor \frac{P_{i+1}}{P_i} \rfloor}{P_{m+1}} \rceil$$

Given knowledge of $a_m, \forall m < i$, this value, $e_{i+1,k}^{spent}$, can be completely calculated as a function of a_i. For simplicity, we define t_i^{avail} to be the minimum amount of time available at priority level i for work of priority level i or lower for the interval $P_i * \lfloor \frac{P_{i+1}}{P_i} \rfloor$.

$t_i^{avail} = P_i * \lfloor \frac{P_{i+1}}{P_i} \rfloor - \sum_{m=1}^{i-1} a_m * \lceil \frac{P_i * \lfloor \frac{P_{i+1}}{P_i} \rfloor}{P_{m+1}} \rceil$

The remaining resource requirement, $e_{i+1,k}^{left}((j-1)*P_i)$, is simply calculated from the assumed original resource requirement and $e_{i+1,k}^{spent}$ as follows.

$$\begin{aligned} e_{i+1,k}^{left}((j-1)*P_i) &= e_{i+1}^{maxp} - e_{i+1,k}^{spent} \\ &= e_{i+1}^{maxp} - t_i^{avail} + a_i \\ &= P_{i+1} - \sum_{m=1}^{i} a_m * \lceil \frac{P_{i+1}}{P_{m+1}} \rceil - t_i^{avail} + a_i \\ &= P_{i+1} - t_i^{avail} - \sum_{m=1}^{i-1} a_m \lceil \frac{P_{i+1}}{P_{m+1}} \rceil \end{aligned}$$

This value, $e_{i+1,k}^{left}((j-1)*P_i)$, can now be used to determine the probability that a job in the first phase will pass the second admission test.

$$\begin{aligned} &P(e_{i+1,k}^{left}((j-1)*P_i) + e_{i,j} \leq e_i^{maxp}) = \\ &\quad P(e_{i,j} \leq e_i^{maxp} - P_{i+1} + t_i^{avail} + \sum_{m=1}^{i-1} a_m \lceil \frac{P_{i+1}}{P_{m+1}}) \end{aligned}$$

The above probability needs to be related to those calculated according to the number of phases. Assuming the worst case—that all jobs in the first phase must undergo this secondary admission test, we get the following.

$$\begin{aligned} &P(S_{i,1} = 1 | e_{i,1} \leq a_i) = \\ &\quad P(e_{i,j} \leq e_i^{maxp} - P_{i+1} + t_i^{avail} + \sum_{m=1}^{i-1} a_m \lceil \frac{P_{i+1}}{P_{m+1}} \rceil) \end{aligned}$$

Theorem 2 *Given a task set with arbitrary periods, the probability than an arbitrary job $\tau_{i,j}$ of task τ_i will be admitted is $QoS(\tau_i)$—the QoS function of τ_i. $QoS(\tau_i)$ can be computed through the calculation of the following values as detailed in [2].*

1. $P(S_{i,1} = 1) = P(e_{i,1} \leq a_i) * P(S_{i,1} = 1 | e_{i,1} \leq a_i)$
2. $\forall k \leq \lceil \frac{P_{i+1}}{P_i} \rceil,\ P(\sum_{n=1}^{k} e_{i,k} \leq a_i)$
3. $\forall k \leq \lceil \frac{P_{i+1}}{P_i} \rceil,\ P(\sum_{n=1}^{k} e_{i,k} > a_i)$

3.3. Calculating Allowances from QoS Requirements

In the previous two subsections, we have shown how to calculate a QoS guarantee for a task given its allowance. However, the reverse operation is necessary. Tasks (message flows) will require a given QoS. The system must determine whether it can support that QoS. If the QoS can be guaranteed, then the task can be admitted to the system and its allowance must be calculated; otherwise the task must be rejected—namely, $a_i^x \leq a_i^y \Rightarrow QoS(\tau_i^x) \leq QoS(\tau_i^y)$.

As expressed above, the transformation from QoS to allowance requires that QoS increases monotonically with increasing allowance. As can be seen from our analysis, this is the case. Therefore, a binary search can be used to find the minimum allowance which satisfies the QoS requirement of a task. For a binary search, the maximum and minumum values must be specified. Obviously, the minimum allowance is zero. The maximum possible allowance is the task's superperiod, guaranteeing 100% utilization for that task.

4. Performance Evaluation

To evaluate the performance of SRMS-based VBR traffic flow multiplexing, and to compare the application QoS it delivers with that it promises through the analytical QoS calculations presented in section 3, we developed a simulator to run a periodic task system subject to the model and assumptions discussed in section 2. Namely, a periodic task system represents a set of VBR traffic flows that are multiplexed on a single CBR channel. Thus, in this section, we use the terms *"task" ("resource")* and *"VBR traffic flow" ("CBR channel")* interchangeably.

We conducted two sets of simulation experiments. In the first, we used task sets consisting of five periodic tasks with harmonic periods. The first period was fixed, and the remaining periods were chosen randomly, so that the ratio between adjacent periods was an integer uniformly distributed between 1 and 4. In the second, we used task sets consisting of five periodic tasks with arbitrary periods, where the ratio between adjacent periods are uniformly distributed between 1.75 and 6.

For comparison purposes, we considered the *Job Failure Rate* (JFR) as our performance metric. JFR is the percent of missed deadlines per task, averaged over all tasks [1]. The use of JFR as a performance metric is superior to the conventional percentage missed deadlines, because JFR gives equal weights to all tasks as opposed to equal weights to all jobs.

In our experiments, we evaluated three versions of SRMS. The first version, which we term *Basic SRMS*, works only on harmonic periods and includes no heuristic optimizations. The harmonic QoS calculations in section 3 were based upon an analysis of Basic SRMS. The second version, which we term *Simple SRMS*, works on arbitrary periods and includes no heuristic optimizations. The non-harmonic QoS calculations in section 3 were based upon an analysis of Simple SRMS. The third version, which we term *SRMS*, is an improvement of Simple SRMS. In particular, it allows unreserved/unclaimed resource utilization to be used to improve the overall system performance in a best-effort fashion (above and beyond the minimal guaranteed QoS).

The first set of experiments considered a constant resource requirement. This removes the statistical nature of the QoS guarantee and allows us to verify our analysis. As can be seen in the leftmost plots in Figures 4 and 5, the JFR based on the calculated QoS for each task matches the JFR simulated with Basic SRMS or Simple SRMS.

We considered exponential, gamma, poisson, normal, uniform and Pareto distributions of resource requirements. This allowed us to determine if the algorithms' behaviors changed based upon distribution. We found that the gross behavior of the algorithms did not vary significantly, and that the QoS analysis maintains its relevance. Therefore, we will only show the results of the poisson and normal distributions.

Our experiments show that the calculated QoS generally provided a good upper bound for Basic SRMS under all distributions considered. Since the calculated QoS (as derived in section 3) is a *statistical* guarantee, it is important to note that, occasionally, in any experiment of finite length,

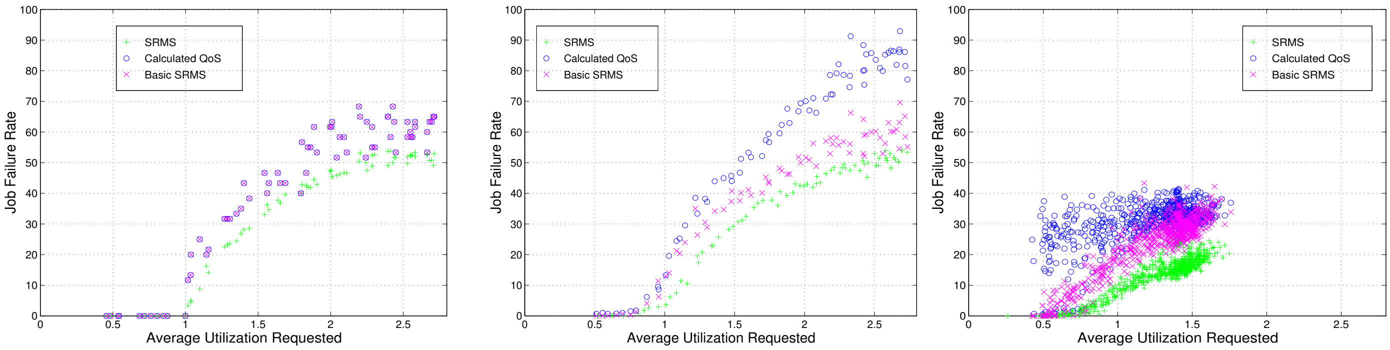

Figure 4. JFR for constant (left), Poisson (middle) and normal (right) message sizes with harmonic periods.

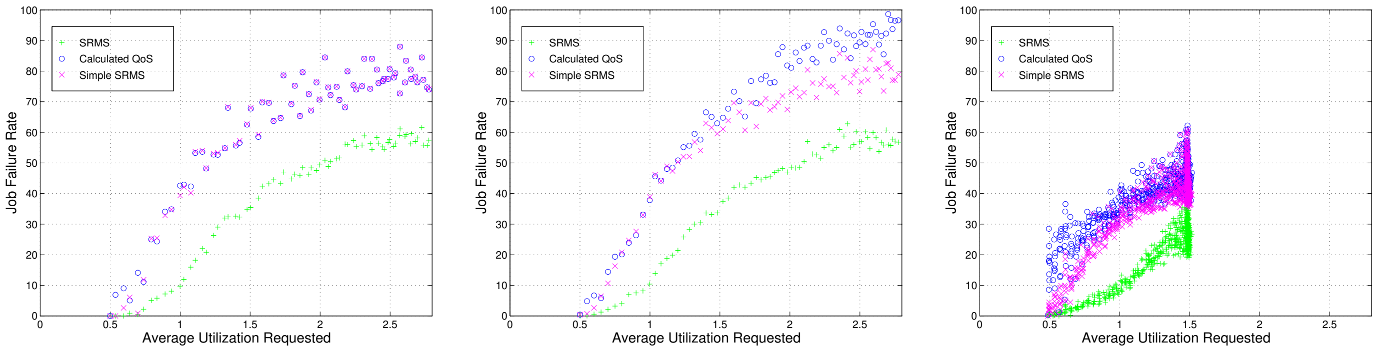

Figure 5. JFR for constant (left), Poisson (middle) and normal (right) message sizes with arbitrary periods.

Basic SRMS may not deliver the promised QoS. This can be seen in the simulations with normal distributions (rightmost plots in Figures 4 and 5, where, rarely, basic SRMS behaves worse than the calculated QoS.[1]

As expected, our experiments show that the QoS delivered using SRMS (with all possible improvements) is far superior than that delivered by Basic SRMS under all distributions. The difference is more pronounced for task sets with arbitrary periods (Figure 5) This is due to the fact that the unutilized/unclaimed resource utilization is "larger" for task sets with arbitrary periods, compared to that for task sets with harmonic periods. The calculated and guaranteed QoS is a statical bound on the QoS provided by the basic SRMS algorithm; with the unanalyzed improvements, the statistical nature of the guarantee is less significant because a QoS superior to that calculated is actually delivered.

5. SRMS Workbench

For demonstration purposes, we have packaged: (1) the SRMS QoS negotiator (i.e. schedulability analyzer), and (2) our SRMS simulator (Basic SRMS + all extensions) into a Java Applet that can be executed remotely on any Java-capable Internet browser. For comparison, a RMS [5] simulator and a SSJAC [3] simulator are included. The SRMS Workbench is available at:

`http://www.cs.bu.edu/groups/realtime/SRMSworkbench`

[1] In some cases, the difference between the calculated QoS and the QoS delivered through Basic SRMS is not negligible. We believe that this is due to the truncation of the probability distributions (in the simulation) and to the randomness of the resource requirements. As evidence, when the resource requirement is fixed (leftmost plots in Figures 4 and 5), the calculated QoS and that obtained through Basic SRMS are identical.

References

[1] A. K. Atlas. *Statistical Rate Monotonic Scheduling: Quality of Service through Management of Variability in Real-time Systems.* PhD thesis, Boston University, June 1999.

[2] A. K. Atlas and A. Bestavros. Multiplexing vbr traffic flows with guaranteed application-level qos using statistical rate monotonic scheduling. Technical Report BUCS-TR-98-011, Boston University, Computer Science Department, 1998.

[3] A. K. Atlas and A. Bestavros. Slack stealing job admission control. Technical Report BUCS-TR-98-009, Boston University, Computer Science Department, 1998.

[4] A. K. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *IEEE Real-Time Systems Symposium*, Dec. 1998.

[5] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973.

[6] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 39:1175–1185, 1990.