

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

Alia K. Atlas and Azer Bestavros

Computer Science Department
Boston University
Boston, MA 02215
{akatlas, best}@cs.bu.edu

Abstract

Quality of Service (QoS) guarantees are required by an increasing number of applications to ensure a minimal level of fidelity in the delivery of application data units through the network. Application-level QoS does not necessarily follow from any transport-level QoS guarantees regarding the delivery of the individual cells (e.g. ATM cells) which comprise the application's data units. The distinction between application-level and transport-level QoS guarantees is due primarily to the fragmentation that occurs when transmitting large application data units (e.g. IP packets, or video frames) using much smaller network cells, whereby the partial delivery of a data unit is useless; and, bandwidth spent to partially transmit the data unit is wasted.

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.

Keywords: ATM Networks; multiplexing of VBR flows; scheduling algorithms; Quality of Service (QoS) management; Real-time computing and communication.

*This work was partially supported by NSF research grant CCR-9706685.

1 Introduction

An important goal of ATM networks is to provide guaranteed Quality of Service (QoS) to the different types of data flows which are carried by the network. For Constant Bit Rate (CBR) traffic, these guarantees are simple to support. However, for Variable Bit Rate (VBR) traffic, QoS is more challenging to guarantee. Multiplexing of different flows is necessary to provide reasonable utilization of the resource and to permit the acceptance of additional data flows. The guarantees supplied by most algorithms are for the individual ATM cells. Applications do not care about the QoS extended to the transport of individual cells. Instead, applications require QoS for their application-level data units, or *messages*.

Motivation: 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. As we discuss in Section 3, the multiplexing of VBR data flows can occur at a given switch and require only a CBR reservation from the remainder of the network.

In [1] we have introduced Statistical Rate Monotonic Scheduling (SRMS)—a algorithm that allows for the efficient scheduling of periodic real-time task systems with statistical QoS guarantees. SRMS proceeds in two phases: a feasibility test and a scheduling algorithm. The feasibility test for SRMS ensures that using the SRMS algorithm, it is possible for a given periodic task set to share a given resource (e.g. processor, communication medium, switching device, etc.) in such a way that such sharing does not result in the violation of any of the periodic tasks QoS constraints. SRMS' schedulability test is simple and its scheduling algorithm has a constant overhead (i.e. the complexity of the scheduler is not dependent on the number of the tasks in the system).

Paper Scope and Outline: 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 remainder of this paper is organized as follows. In section 2, we present previous work on resource management and scheduling in networks. In section 3, we introduce a network model which uses SRMS at the border switches. In subsection 3.1, we review the SRMS task model and scheduling algorithm. In section 4, we present our SRMS-based paradigm for the management of QoS constraints. In section 5, we present the results of simulations using SRMS to demonstrate the accuracy of our analysis. In section 6, we overview the SRMS Workbench, a Java-based Web application that enables interactive specification, schedulability analysis, QoS negotiation, and simulation of task sets under SRMS. We conclude in section 7 with a summary and directions for future research.

2 Related Work

The problem of scheduling multiple data streams across a single link has been extensively studied. The most common and popular method, Weighted Fair Queueing (WFQ) [2] (also known as Packetized Generalized Processor Sharing (PGPS) [3]) uses proportional shares to distribute the outlink’s bandwidth to the competing streams. An analysis of PGPS yields QoS for delay, jitter and other characteristics [3, 4]. Self Clocked Fair Queueing [5, 6] was designed to reduce the computational complexity of WFQ. Start-time Fair Queueing [7] attempts to achieve fairness regardless of server capacity variation. The advantage of fair scheduling algorithms is that they guarantee that bandwidth will be fairly allocated, regardless of prior use or current overload.

A different approach, which still supports fair scheduling, is traffic-controlled rate monotonic priority scheduling [8]. This approach is used to schedule ATM cells. Each data stream has a leaky bucket model, and at each switch, the traffic is controlled so that it remains compliant throughout the system. A period is assigned to each data stream which is the inverse of the rate of cells which that data stream can generate. Thus, each period, each stream requires the transmission of one ATM cell. With these periods, the traffic is scheduled according to rate monotonic scheduling (RMS) [9]. This algorithm is introduced to multiplex streams to provide bounded delays for real-time communication services.

As we mentioned in the introduction, QoS guarantees are required by the application, which is oblivious to the QoS extended to network cells. This consideration is discussed in [10, 11, 12]. To schedule VBR video traffic, the idea of “burst scheduling” was presented in [10, 11]. Essentially, an application transmits video frames at a fixed rate, but—because of the varying frame size—the rate of ATM cells (and thus the bandwidth required) varies. Therefore, when a new frame is to be transmitted, an attempt is made to reserve the necessary resources for that frame’s required bandwidth. If the reservation cannot be guaranteed, the frame is dropped.

In [12], packets or cells are grouped together and associated with one deadline. In the servers which are considered, packets in a single flow are transmitted FIFO; therefore order is preserved. Assigning the same deadline to multiple packets simplifies algorithms which dynamically assign priorities based upon deadlines. Because the application is concerned only with the end-to-end delay of the entire application data unit, all packets in that data unit can have the same deadline as the last packet in the data unit.

The fragmentation of IP packets when carried over ATM networks is another instance where the distinction between network cells and application data units is evident. Sending an IP packet over ATM requires fragmenting the IP packet into a number of ATM cells, transmitting these cells over the ATM network, and reassembling the IP packet from its constituent cells. Cells of IP packets are usually transmitted at ABR or UBR. Dropping a single cell results in bandwidth wasted in transmitting the rest of the packet. There have been a number of attempts to remedy this problem by introducing additional switch-level functionalities to preserve throughput when IP is employed over ATM. Examples include the Selective Cell Discard (SCD)¹ [14], Early Packet Discard (EPD) [13], and the Lazy Packet Discard (LPD) employed in TCP-Boston [15].

As mentioned in [10, 11], similar difficulty (resulting from the fragmentation of data units into cells) is encountered when transmitting video frames or other application-specific data units as VBR traffic. In [16, 17], a dynamic priority technique is proposed to attempt to guarantee (m,k)-hard deadlines.²

¹Also called Partial Packet Discard (PPD) in [13].

²In [18, 19], to deal with overload when scheduling using RMS, the idea of skipping some instances of periodic tasks (i.e. jobs) was introduced and generalized to define (m,k)-hard deadlines.

Essentially, each data stream is modeled as a stream of customers, with specific deadlines and varying resource requirements. Out of any k consecutive customers, at least m must meet their deadlines. Each stream is assigned a priority, based upon how close that stream is to violating its (m,k) constraint. Within the same priority level, the head customer from each stream is scheduled according to Earliest Deadline First [9]. The probability of violating the (m,k) guarantee, suffering a dynamic failure, was determined in [17].

In [20], the above algorithm was modified to schedule MPEG video streams so that more frames (customers) would meet their deadlines. Essentially, a frame was marked as *urgent*, if it was an I-frame or if the frame missing its deadline would result in dynamic failure. Then Earliest Deadline as Late as possible (EDL) was used to schedule a specified number of urgent frames from each stream. Non-urgent frames are scheduled until an pre-scheduled urgent frame must run.

Both EDF, a dynamic priority algorithm, and RMS, a fixed priority algorithm, have their roots in real-time scheduling theory [9]. RMS, in particular, requires periodic task sets with constant resource requirements. For a given task system, if the RMS schedulability test is passed, then the task system is guaranteed to meet every deadline.³ For this guarantee, the utilization requirements of the task set must be fixed. This would work well for CBR traffic, but there is no need to do statistical multiplexing of CBR traffic; a fair proportional share scheduling algorithm works.

VBR traffic has variable utilization requirements. RMS could be used to schedule periodic task sets with variable resource requirements, but deadlines will be missed. To determine a task's QoS in this instance, Tia *et al.* in [21] introduced a probabilistic time-demand analysis, which yields the probability of making a given task's deadline under RMS. The authors also introduced a transform-task method, and provided a probabilistic guarantee for longer jobs and an absolute guarantee for short jobs. In their latter work, the authors assumed that all work must be completed, even if it was late. As previously discussed, in the context of transmitting application data units, this simply wastes bandwidth.

3 Network Model and SRMS Framework

Our network model in this research consisted 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 border switch is connected to the larger network, with a guaranteed CBR connection to all other border switches which it has traffic going to. In this model, we assume that the bandwidth bottlenecks exist in the main network, not in the internal networks. Therefore, we assume that no multiplexing is necessary until the border switch.

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

³EDF can provide similar guarantees.

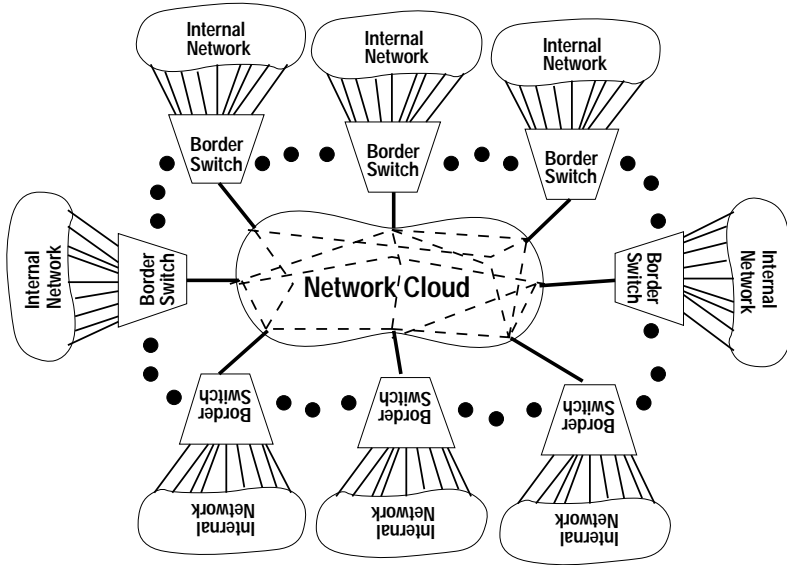


Figure 1: Model of Network

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, an application's message traffic resembles a classical real-time periodic task model, with two differences. 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 is provided. We will consider each message flow to represent a periodic task in the rest of the paper. To solve a similar problem, we introduced SRMS [1].

3.1 Statistical Rate Monotonic Scheduling

Definition 1 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.

Definition 2 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 .

Period Transformation for Preemptive Scheduling 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 $P_i = \lfloor \frac{1}{CT} \rfloor$. Therefore, all preemptions occur at the

end of a cell’s transmission.

Rate Monotonicity: 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.

Job Ready Time and Deadlines: 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 j th 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 $j * P$.

Resource Requirements, Allowances, and Schedulability: 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), $f(x)$. Obviously, it is impossible for a job to require more than 100% of the resource. Thus, $x > P \rightsquigarrow f(x) = 0$. 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.⁴ The resource requirement for the j th job of the i th 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 allotted 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 $QoS(\tau_i)$.

Definition 3 *The superperiod of τ_i is P_{i+1} , the period of the next lower priority task, τ_{i+1} .*

Definition 4 *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.

Definition 5 *A set of tasks $\tau_1, \tau_2, \dots, \tau_n$ 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.*

Overview of SRMS Algorithm 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

⁴This assumption allows SRMS to test a job for admittance and to guarantee that all admitted jobs will meet their deadlines.

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 [1].

4 Quality of Service 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 4.2, we present a generalization for task systems with arbitrary periods. Finally, we will discuss calculating the allowance from QoS requirement.

4.1 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 [1] 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 $\tau_1, \tau_2, \dots, \tau_n$, 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 the above lemma follows directly 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 separate 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.

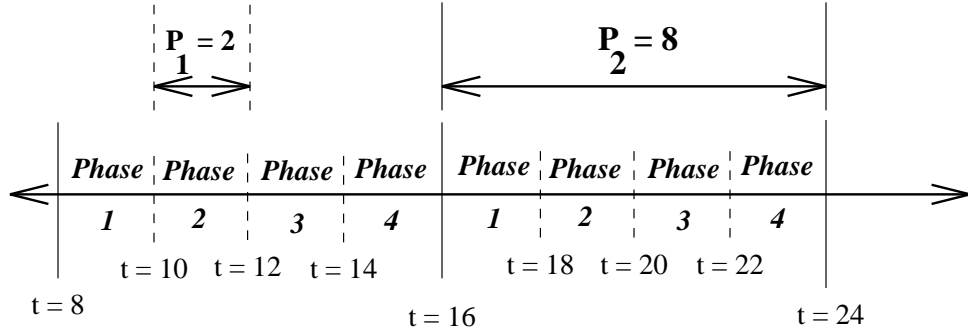


Figure 2: Sample Task with Four Phases

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 k th 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.

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

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 .*

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

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 System with 4 Tasks and Maximum Utilization 1.178

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.

Example Calculations To give some concrete feel for the above formulas, consider the example task system given in Table 1, which lists the period and requested resource utilization (as a uniform distribution with a known mean and maximum) for each task. Also shown is the number of phases for each task (i.e. $\frac{P_{i+1}}{P_i}$).

The results of applying the formulas given in Theorem 1 are seen in the three tables shown in figure 2. The $P(S_{i,j} = 1)$ headings are the probability that an arbitrary job $\tau_{i,j}$ will meet its deadline.

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:

$$\sum_{i=1}^4 \frac{a_i}{P_{i+1}} \leq 1$$

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.

To illustrate the behavior of SRMS, Figure 3 shows one of the possible schedules resulting from the fourth resource assignment in Table 3—namely ($a_1 = 4$, $a_2 = 6$, $a_3 = 33$, $a_4 = 3$).

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

In Figure 4, 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 [1]. As can also be seen in Figure 4, 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

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: QoS Calculations for Example Task System

a_1	a_2	a_3	a_4	Utilization	$QoS(\tau_1)$	$QoS(\tau_2)$	$QoS(\tau_3)$	$QoS(\tau_4)$
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 with QoS

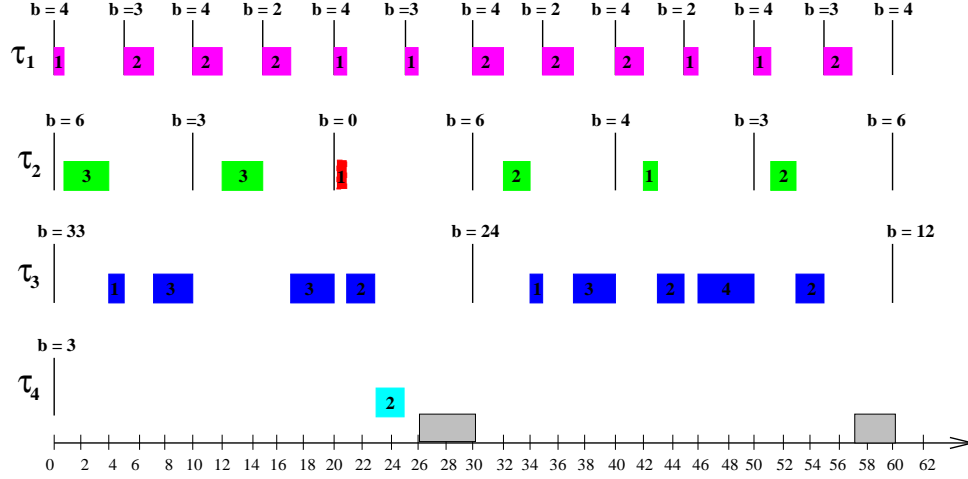


Figure 3: Possible Schedule of Example Task System

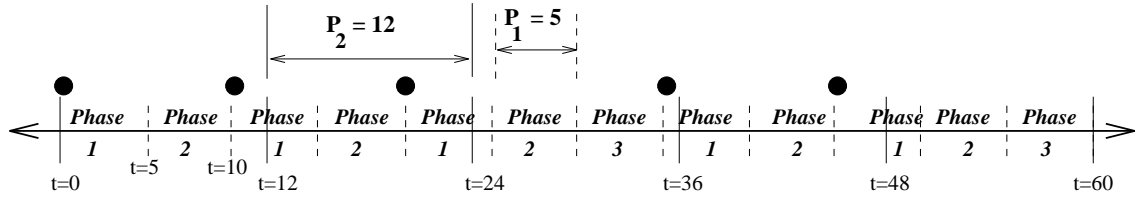


Figure 4: Phases for Task with Overlap Jobs

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.

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

$LCM(P_{i+1}, P_i)$ 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.⁵ Thus, the probability that a job will be able to meet its deadline (i.e. $P(S_{i,k} = 1)$) is equal

⁵This is identical to the admission test for the harmonic task sets discussed before.

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 $P(e_{i,1} \leq a_i)$ ($P(e_{i,1} > a_i)$) 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 j th 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 k th phase meets its deadline.

The probability that a job in the first phase will pass the second admission test still remains to be calculated. To calculate it, 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 [22]. 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. It is given by:

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

Definition 6 *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 $P(e_{i+1,k}^{left}((j-1)*P_i) + e_{i,j} \leq e_i^{maxp})$. 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 j th job is released, the minimum resource requirement spent on the k th job of task τ_{i+1} is given by:

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

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 $P_i * \lceil \frac{P_{i+1}}{P_i} \rceil$.

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

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.

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

The above probability needs to be related to the probabilities 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.

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

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

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 shown in Appendix A.*

1. $P(S_{i,1} = 1) = P(e_{i,1} \leq a_i) * P(e_{i,j} \leq e_i^{maxp} - P_{i+1} + t_i^{avail} + \sum_{k=1}^{i-1} a_k \lceil \frac{P_{i+1}}{P_{k+1}} \rceil)$
2. $\forall k \leq \lceil \frac{P_{i+1}}{P_i} \rceil, P(\sum_{n=1}^k e_{i,n} \leq a_i)$
3. $\forall k \leq \lceil \frac{P_{i+1}}{P_i} \rceil, P(\sum_{n=1}^k e_{i,n} > a_i)$

4.3 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.

$$a_i^x \leq a_i^y \Rightarrow QoS(\tau_i^x) \leq QoS(\tau_i^y)$$

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 minimum 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.

For non-harmonic task sets, the QoS provided by an allowance depends upon the allowance of higher priority tasks. Therefore, the QoS calculation must be done in rate monotonic order for the tasks. If all tasks cannot be guaranteed their desired QoS, the least critical task or the task applying for admittance can be rejected, and the calculations performed again. This allowance calculation algorithm has been implemented in the SRMS Workbench.

5 Evaluation of SRMS Analysis

To evaluate the performance of SRMS in general, and to compare the QoS it delivers with that it promises through the analytical QoS calculations presented in section 4, we developed a simulator to run a periodic task system subject to the model and assumptions discussed in section 3.

Simulation Model: In our experiments, we made a number of simplifying assumptions. These assumptions were necessary to allow for a more straightforward interpretation of the simulation results, by eliminating conditions or factors that are not of paramount interest to the subject matter of this paper (e.g. effects of task criticality). First, we assumed that all tasks demand the same average percentage utilization of the resource being managed. In other words, the ratio $\frac{E(e_{i,j})}{P_i}$ for all tasks is constant. Second, the probability distributions used to generate the resource requirements were of the same type⁶ (but with different parameters) for each task in the system. Also, these distributions were truncated so that no infeasible jobs were submitted to the system (note that the probability distributions were not truncated for the analytical QoS analysis of section 4). Third, we assumed that all tasks were of equal criticality/importance, which implies that the assignment of allowances (a_1, a_2, \dots) to the tasks in the system should not reflect any preferability due to the task’s “value” to the system.

We conducted two sets of simulation experiments. In the first set of experiments, we used task sets consisting of five periodic tasks⁷ 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 one and four. For our second set of simulation experiments, we used task sets consisting of five periodic tasks with arbitrary periods. The first period was fixed, and the remaining periods were randomly chosen, with the ratio between adjacent periods uniformly distributed between 1.75 and six.

⁶We considered a variety of such distributions as will be evident later in this section.

⁷The small size of our task sets was chosen to permit comparison against an *optimal oracle* [1].

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. Assuming that there are N tasks, the formula for JFR is given in equation 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.⁸ We compare JFR against the average requested utilization of the system. For simplicity, we have chosen to graph the calculated QoS against the requested utilization for the unoptimized SRMS simulation; this properly associates the QoS guarantee with the simulation of the same system.

$$JFR = \frac{1}{N} * \sum_{i=1}^N \frac{\text{missed jobs of } \tau_i}{\text{all jobs of } \tau_i} \quad (1)$$

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 4 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 4 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).

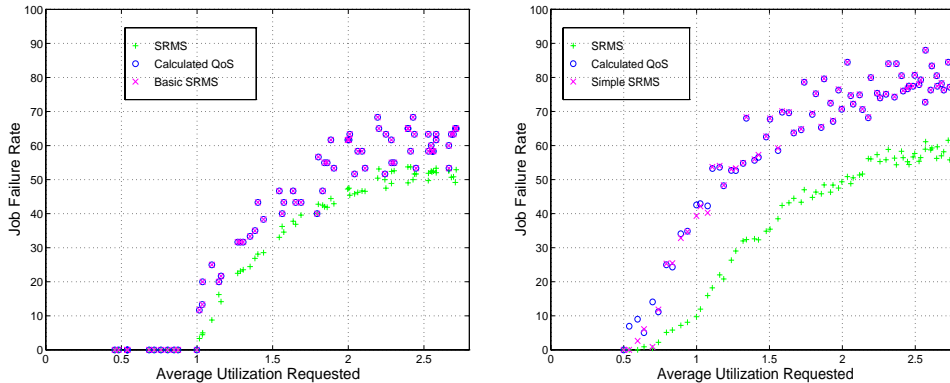


Figure 5: Job Failure Rate for constant message sizes with harmonic (left) and arbitrary (right) periods.

In Figure 5, we 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 Figure 5, the JFR based on the calculated QoS for each task matches the JFR simulated with Basic SRMS or Simple SRMS.

In addition to a fixed PDF, we considered exponential, gamma, poisson, normal, uniform and pareto distributions. 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.

⁸Specifically, the percentage of missed deadlines is biased towards tasks with shorter periods, since those tasks have more deadlines in a given interval than tasks with longer periods.

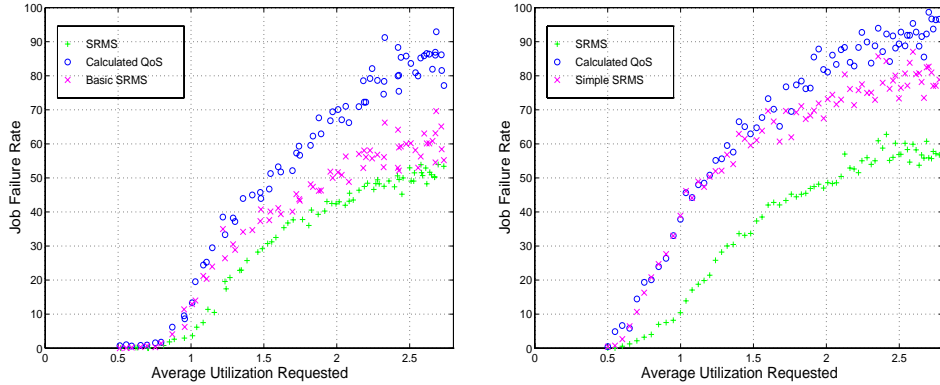


Figure 6: Job Failure Rate for poisson-distributed message sizes with harmonic (left) and arbitrary (right) periods.

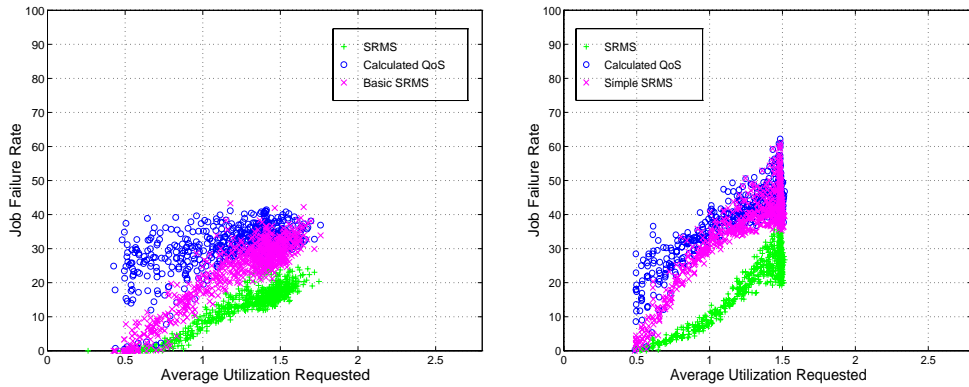


Figure 7: Job Failure Rate for normally-distributed message sizes with harmonic (left) and arbitrary (right) periods.

We choose to use these two distributions because they can represent the PDF of some realistic applications. For instance, the poisson distribution was frequently used to model network traffic.⁹ Therefore, we wished show results using this PDF. The normal distribution is of great interest, because it is used to model many phenomena and because the sum of i.i.d. random variables converge to a normal distribution.

Simulation Results: 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 4) is a *statistical* guarantee, it is important to note that, occasionally, in any experiment of finite length, Basic SRMS may not deliver the promised QoS. This can be seen in the simulations with normal distributions (figure 5, where, rarely, basic SRMS behaves worse than the calculated QoS. In some cases, the difference between the calculated QoS and the QoS delivered through Basic SRMS is not negligible. We believe that

⁹The issues of self-similarity in network traffic and in file size of file systems have caused increased interested in heavy-tailed distributions.

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 (figure 5), the calculated QoS and that obtained through Basic SRMS are identical.

For this set of experiments, we used an analytical form of the PDFs to calculate the QoS. As implemented in the SRMS workbench[23], the QoS calculation are based upon an array of sample resource requirements. The latter approach permits easy calculation of truncated distributions. However, there is a potential inaccuracy in the PDF, since it is only represented by samples.

As expected, our experiments show that the QoS delivered using SRMS (with all possible improvements) is far superior than that delivered by Basic SRMS (for harmonic periodical task sets) and to that delivered by Simple SRMS (for non-harmonic periodical task sets) under all distributions. The difference is more pronounced for task sets with arbitrary periods (all plots on the right-hand-side of figures 5-5). 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.

6 SRMS Workbench

For demonstration purposes, we have packaged: (1) the SRMS schedulability analyzer (QoS negotiator), 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 [9] simulator and a SSJAC [24] simulator are included.

Through a simple GUI, the *SRMS Workbench* allows users to specify a set of periodic tasks, each with (a) its own period, (b) the distributional characteristics of its periodic resource requirements (e.g. Poisson, Pareto, Normal, Exponential, Gamma, etc.), (c) its desired QoS as a lower bound on the percentage of deadlines to be met, and (d) a criticality/importance index indicating the value of the task (relative to other tasks in the task set). Once the task set is specified, the SRMS Workbench allows the user to check for schedulability under SRMS. If the task set is schedulable, the SRMS Workbench generates the appropriate allowance for each task and allows the user to create an animated simulation of the task system, which can be executed and profiled. If the task set is not schedulable, the SRMS Workbench informs the user of that fact and suggest (as part of the QoS negotiation) an alternative set of *feasible* QoS requirements that reflects the specified criticality/importance index of the tasks in the task set.

The SRMS Workbench is available at: <http://www.cs.bu.edu/groups/realtime/SRMSworkbench>

7 Conclusion

In this paper, we presented a new paradigm for multiplexing VBR traffic flows with guaranteed QoS using SRMS [1]. We considered applications which generate messages at a constant rate, but where the message size is variable. The QoS required is that a given percentage of the messages are successfully transmitted; partial transmission of messages is useless.

The advantages of our scheme are as follows:

- (1) The assignment of tasks to resources is done using a low-overhead *fixed-priority* scheduling algorithm, which makes it quite attractive for on-line QoS management.
- (2) The QoS extended to various tasks is *adjustable* and independent of task periods. This allows the system to treat tasks preferentially to reflect their relative *criticality/value* in total isolation of the task periods. Also, this allows performance to degrade gracefully and predictably under conditions of overload.
- (3) Admission control decisions are made as soon as jobs are released. This allows for early rejections of jobs that are not guaranteed to meet their deadlines, and consequently preserving resources.
- (4) The *firewall* property is enforced in SRMS, so that two message flows cannot adversely affect each other.
- (5) SRMS QoS management allows the integration of reservation-based and best-effort resource scheduling seamlessly.

Future Work We would like to implement our algorithm in an ATM switch with the network configuration described. Then we could experiment with message flows from real applications. We plan to consider how to modify this analysis so that it could apply to networks with variable packet size, such as IP.

The QoS presented in this paper does not include any guarantees upon the message delay. A further analysis of the system is necessary to determine what delay guarantees can be made for the messages which are successfully transmitted. As with RMS, the guaranteed utilization which SRMS can support is approximately 69%. Extending SRMS to support dual priorities [25] would enable full utilization to be guaranteed. Additionally, the analysis and algorithm given assume a fixed set of tasks. We plan to consider extending the algorithm to support the addition and deletion of additional tasks. Such mode changes have been considered for RMS [26, 27] and could be generalized to SRMS. There is also future work in QoS negotiation middleware.

In our current model, we assume CBR connections or virtual circuits between the border switches are already extant. We would like to consider schemes where the bandwidth available can be negotiated, over a longer timespan, based upon the number of rejected connections and the requested utilization. Therefore, as demand varies during the day, the bandwidth available to serve that demand will vary.

References

- [1] Alia Atlas and Azer Bestavros. Statistical rate monotonic scheduling. Technical Report BUCS-TR-98-010, Boston University, Computer Science Department, May 1998. Accepted for publication in IEEE Real-Time Systems Symposium, December 1998, Madrid, Spain.
- [2] A. Demers, S. Keshav, and S. Shenkar. Analysis and simulation of a fair queueing algorithm. In *Proceedings of ACM SIGCOMM*, pages 1–12, Sept 1989.
- [3] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated service networks: The single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, Jun 1993.
- [4] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated service networks: The multiple node case. In *Proceedings of IEEE INFOCOM*, pages 521–530, March 1993.
- [5] J. Davin and A. Heybey. A simulation study of fair queueing and policy enforcement. *Computer Communication Review*, 20(5):23–29, Oct 1990.
- [6] S. J. Golestani. A self-clocked fair queueing scheme for high speed applications. In *Proceedings of IEEE INFOCOM*, 1994.
- [7] Pawan Goyal, Harrick M. Vin, and Haichen Cheng. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks. In *Proceedings of ACM SIGCOMM*, 1996.
- [8] Seok-Kyu Kweon and Kang G. Shin. Traffic-controlled rate-monotonic priority scheduling. In *Proceedings of IEEE INFOCOM*, pages 655–662, 1996.
- [9] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973.
- [10] Simon S. Lam and Geoffrey G. Xie. Burst scheduling: Architecture and algorithm for switching packet video. In *Proceedings of IEEE Infocom'95*, 1995.
- [11] Simon S. Lam and Geoffrey G. Xie. Burst scheduling networks. Technical Report TR-94-20, Department of Computer Science, University of Texas at Austin, Oct 1996. Third revision.
- [12] Simon S. Lam and Geoffrey G. Xie. Group priority scheduling. In *Proceedings of IEEE Infocom'96*, 1996.
- [13] Allyn Romanow and Sally Floyd. Dynamics of TCP Traffic over ATM Networks. *IEEE Journal on Selected Areas in Communications*, 13(4):633–641, May 1995.
- [14] G. Armitage and K. Adams. Packet Reassembly During Cell Loss. *IEEE Network Mag.*, 7(5):26–34, September 1993.
- [15] Azer Bestavros and Gitae Kim. TCP Boston: A Fragmentation-tolerant TCP Protocol for ATM Networks. In *Proceedings of Infocom'97: The IEEE International Conference on Computer Communication*, Kobe, Japan, April 1997.
- [16] Moncef Hamdaoui and Parameswaran Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transactions on Computers*, 44(12):1443–1451, Dec 1995.

- [17] Moncef Hamdaoui and Parameswaran Ramanathan. Evaluating dynamic failure probability for streams with (m,k)-firm deadlines. *IEEE Transactions on Computers*, 46(12):1325–1337, Dec 1997.
- [18] Gilad Koren and Dennis Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *Real-Time Systems Symposium*, 1995.
- [19] Guillem Bernat and Alan Burns. Combining (n, m)-hard deadlines and dual priority scheduling. In *Real-Time Systems Symposium*, pages 46–57, 1997.
- [20] Ching-Chih Han and Kang G. Shin. Scheduling mpeg-compressed video streams with firm deadline constraints. In *Proceedings of the Third ACM International Multimedia Conference and Exhibition*, pages 411–422, Nov 1995.
- [21] T.S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu. Probabilistic performance guarantees for real-time tasks with varying computation times. In *Real-Time Technology and Applications Symposium*, pages 164–173, May 1995.
- [22] 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.
- [23] Alia Atlas and Azer Bestavros. The statistical rate monotonic scheduling workbench. Technical Report BUCS-TR-98-012, Boston University, Computer Science Department, May 1998.
- [24] Alia Atlas and Azer Bestavros. Slack stealing job admission control. Technical Report BUCS-TR-98-009, Boston University, Computer Science Department, 1998.
- [25] A. Burns and A. J. Welling. Dual priority assignment: A practical method for increasing processor utilization. In *Fifth Euromicro Workshop on Real-Time Systems*, pages 48–55, 1993.
- [26] Lui Sha, Rangunathan Rajkumar, John Lehoczky, and Krithi Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Journal of Real-Time Systems*, 1:243–264, 1989.
- [27] K. W. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *Real-Time Systems Symposium*, pages 100–109, Dec 1992.

A Algorithm to Calculate the QoS given in Theorem 2

CalculateQoS($(\forall j \leq i, a_j), (\forall k \leq i + 1, P_k), (\forall k \leq \lceil \frac{P_{i+1}}{P_i} \rceil), P(\sum_{n=1}^k e_{i,k} \leq a_i)$)

$$highCount \Leftarrow \frac{LCM(P_{i+1}, P_i)}{P_i} - \frac{LCM(P_{i+1}, P_i)}{P_{i+1}} * \lfloor \frac{P_{i+1}}{P_i} \rfloor$$

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

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

$$e_{i+1}^{left} = P_{i+1} - t_i^{avail} - \sum_{k=1}^{i-1} a_k \lceil \frac{P_{i+1}}{P_{k+1}} \rceil$$

$$firstMet = P(e_{i,1} \leq a_i) * P(e_{i,1} \leq e_i^{maxp} - e_{i+1}^{left})$$

$$\forall k \leq \lceil \frac{P_{i+1}}{P_i} \rceil, metTable[k] \Leftarrow P(\sum_{n=1}^k e_{i,k} \leq a_i)$$

$$QoS \Leftarrow 0$$

for($phase \Leftarrow 1; phase \leq \lceil \frac{P_{i+1}}{P_i} \rceil; phase ++$)

$$history \Leftarrow 2^{phase-1}$$

$$phaseProb \Leftarrow 0$$

while ($history < 2^{phase}$)

$$missed \Leftarrow 0$$

$$probability \Leftarrow 1$$

for ($j \Leftarrow 1; j \leq phase; j ++$)

if ($history \&\& 2^{j-1} = 1$)

if ($j = 1$)

$$probability \Leftarrow probability * firstMet$$

$$\text{else } probability \Leftarrow probability * metTable[j - missed]$$

else

if ($j = 1$)

$$probability \Leftarrow probability * (1 - firstMet)$$

else

$$probability \Leftarrow probability * (1 - metTable[j - missed])$$

$$missed \Leftarrow missed + 1$$

$$phaseProb \Leftarrow phaseProb + probability$$

$$history \Leftarrow history + 1$$

if ($phase \neq \lceil \frac{P_{i+1}}{P_i} \rceil$)

$$QoS \Leftarrow QoS + phaseProb *$$

$$((highCount / ((highCount + lowCount) * \lceil \frac{P_{i+1}}{P_i} \rceil)) +$$

$$(lowCount / ((highCount + lowCount) * \lfloor \frac{P_{i+1}}{P_i} \rfloor)))$$

else $QoS \Leftarrow QoS +$

$$phaseProb * (highCount / ((highCount + lowCount) * \lceil \frac{P_{i+1}}{P_i} \rceil))$$

return QoS