PROTOCOL AND REAL-TIME SCHEDULING ISSUES FOR MULTIMEDIA APPLICATIONS

A Dissertation Presented

by

SRIDHAR PINGALI

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 1994

Department of Electrical and Computer Engineering

PROTOCOL AND REAL-TIME SCHEDULING ISSUES FOR MULTIMEDIA APPLICATIONS

A Dissertation Presented

by

SRIDHAR PINGALI

Approved as to style and content by:

James F. Kurose, Chair

Donald F. Towsley, Member

Christos G. Cassandras, Member

C. Mani Krishna, Member

Keith R. Carver, Department Chair
Electrical and Computer Engineering

To my parents.

Acknowledgements

It is with deep appreciation that I acknowledge the immense help of my advisors, Professor Jim Kurose and Professor Don Towsley. Their deep insight into the subject matter, their approachability and their willingness to provide funding are what have made this dissertation possible. I would also like to thank them for their careful overseeing of the preparation of this document. Few advisors could have as much concern as they do for the adequate preparation of their students for the "real world". I have learnt enormously from them not just how to do good research, but also how to communicate one's ideas to others well.

I would like to thank Professor Christos Cassandras for his great help in keeping my doctoral studies on track through his impeccable service on all my committees. His attentiveness to my concerns and sense of fairness are gratefully acknowledged.

Professor Mani Krishna has been a source of inspiration and help– his excellence in teaching and willingness to be of assistance have evoked much admiration in me. I thank him for agreeing to serve on my doctoral committee.

I have experienced much joy in being a part of the greater Amherst community. The cultural diversity and spiritual depth of the area have made much personal growth possible, and enabled me to spend a most enjoyable five years in the locality. Many are the friends that I have made, and I bid peace unto them all. In the end, it is this that has made it all worthwhile.

Abstract

PROTOCOL AND REAL-TIME SCHEDULING ISSUES FOR MULTIMEDIA APPLICATIONS

September 1994

Sridhar Pingali, B.Tech., Indian Institute of Technology

M.S., Clemson University

Ph.D., University of Massachusetts Amherst

Directed by: Professor James F. Kurose

The growth of higher bandwidth networks and powerful new workstations has enabled the development of many new multimedia applications. These applications involve the combined use of different media such as voice, video and text. Supporting these applications requires the resolution of a number of issues both within the network and at the end-hosts that originate the applications. In this dissertation we consider real-time scheduling and protocol problems that arise in this arena.

In the realm of applications operating under hard time-constraints, we address both network-node and workstation scheduling. In applications involving the use of video or a combination of voice and video, there can be the need to schedule classes of traffic with differing importance but identical deadlines at a network-node. We examine scheduling algorithms that can be used in the situation of two classes of traffic. We demonstrate using probabilistic arguments that a new "balancing" discipline can provide better loss performance for both classes than more traditional schemes in some operating regions. When delay loss probabilities are low, strict priority scheduling is shown to be a reasonable option.

Turning our attention to the end-hosts, we consider the need to process multimedia objects such as voice packets in a periodic fashion within a workstation, and study

two common scheduling algorithms - earliest deadline first (ED) and rate monotonic (RM). Recent studies have revealed the importance of the preemption behavior of the scheduler in determining overall processor performance. We prove that there are always fewer preemptions under ED than under RM and show through simulations of well-known task-sets that, on occasion, there can be over 20% more preemptions under RM than under ED.

Apart from these scheduling problems, we also look at protocol issues that combine network and host considerations. For applications such as world-wide multimedia lectures, it becomes important to have error-recovery protocols that will scale well to hundreds of receivers. We perform throughput analyses and demonstrate why receiver-initiated protocols based on negative acknowledgements are to be preferred over sender-initiated protocols based on positive acknowledgments. We focus on host processing costs to illustrate this result.

TABLE OF CONTENTS

List of Tables

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The development of higher bandwidth networks and powerful new workstations has opened the doors to a wide range of new multimedia applications such as distributed conferencing and video telephony [1]. These applications use media such as voice, video and shared whiteboard to provide multisensory communication and computing services to users. In designing suitable protocols and scheduling disciplines to provide an acceptable quality of service for such applications, due attention must be given to the various network and computing resources used. The needs of the users and the nature of the applications determine what level of service is adequate. Some applications (e.g., those involving the transfer of medical information) demand *fully reliable* service– i.e, all the data generated at a source must be correctly transferred to the destination. In some other applications such as voice transfer, users are able to tolerate a certain amount of missing data without a significant degradation in the quality of service.

Data can be lost due to a variety of reasons. Traditional network applications lose data as a result of transmission errors, noisy channels and buffer overflows. Another type of loss can occur in multimedia applications when the data have associated "real-time" constraints– i.e., if a message containing data is not available at a destination or has not been processed within a specified amount of time, it is considered to be lost. Henceforth, we shall refer to any loss due to the real-time nature of the data as *delay loss*. These kinds of real-time constraints are especially common in interactive applications such as voice or video where a specified amount of delay loss is tolerable.

Metrics such as the time taken to transfer information from one user of a multimedia application to another (called the end-to-end delay), and the amount of data

lost, characterize the overall quality of service (QOS) provided to the users. Typically, requirements are imposed on the possible values that these performance criteria may take so that an acceptable QOS is delivered to the users. The actual limits on such QOS metrics tend to depend on the characteristics of the application, the nature of the media used, and the purpose of the communication. Figure 1.1 shows examples of different applications on multimedia workstations in a networked environment. Different media are made use of with applications such as NeVot [2] and n.v. [3], which are audio and video conferencing tools respectively, and w.b. [4], which is a distributed and shared drawing environment.

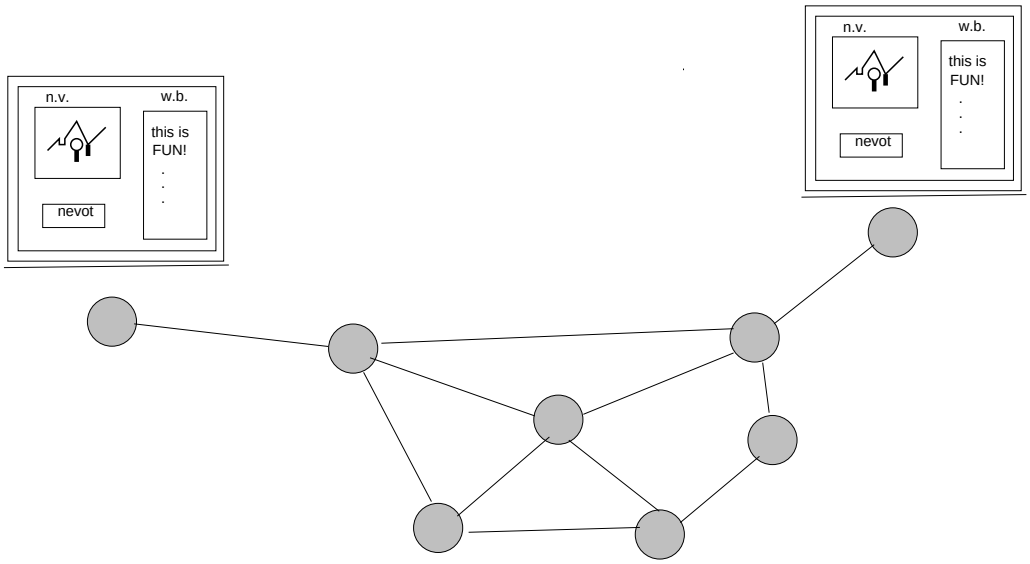

Figure 1.1. Multimedia applications

In order to meet the QOS requirements for any application, computing and network resources are needed. For example, NeVot generates and places voice data at regular intervals on the network link connecting the sending user to the receiving user. For voice data to be in a form that the network can recognize, it has to be processed and encapsulated and, for this, computing resources have to be used at the sending workstation. At the receiving end, the CPU has to strip all network-related

information away from the incoming data and present it to the user in a form that is recognizable as human speech. Resources are also used within the network. Some of the data carrying capacity of the network links is used to transfer this information, and the various nodes within the network must ensure that the voice data is properly routed to the correct destination. It may also be necessary to store data temporarily either at nodes within the network or at the sending and receiving workstations. This uses buffer space. There must be adequate availability of all these resources– CPU power, network bandwidth, buffer space– in order for there to be an acceptable quality to the communication between users.

Another example of resource use arises in applications that require fully reliable service. For distributed conferencing applications using w.b., no data may be lost in communicating between sender and receiver. It is thus necessary that protocols be defined that enable the application to recover from any data losses. This imposes additional requirements on the computational resources at the end-hosts supporting the application.

Providing the necessary quality of service to users becomes a challenging problem when there are *competing* requests for resources. For example, during a multimedia conference, voice and video data may be simultaneously generated by the sender and traverse the same path through the network to a receiver. Since there are end-to-end delay criteria to be met for both voice and video data, there could be contention for CPU resources and for the network links. In such a situation, policies have to be defined to arbitrate between these contending claims and to allocate the available resources appropriately. These resource-allocation policies should consider the rival performance criteria, such as respective time constraints and tolerable delay losses, of the different streams of data. The solutions arrived at in this constrained resource environment must be able to provide an acceptable quality of service to the users.

In this dissertation we study three resource-allocation problems. We look at how to define algorithms and protocols within a network, at a workstation at the edge of

a network, and between peers that communicate over a network, so that proper use may be made of network bandwidth and host processing power. The problems that we consider are:

- Allocation of link bandwidth to achieve performance tradeoffs for two classes of real-time traffic (e.g., voice and video) at a multiplexing point in the network. Here our focus is on comparing the delay loss performance of two classes of traffic under three different link-level packet-scheduling algorithms to determine which algorithm is the best choice.

- CPU allocation for periodic task-sets (e.g., host protocol processing for voice and video). Here we compare the preemption performance of two well-known processor scheduling algorithms to determine which is to be preferred for workstation scheduling.

- Specification of scalable error recovery protocols for multicast applications (e.g., distributed conferencing using a shared whiteboard). Our concern here is to provide fully reliable service by distributing host processing costs so that many hundreds of receivers can be simultaneously supported. We compare the performance of three different protocols to determine which most adequately meets our needs.

We now introduce and motivate each of these problems in turn.

1.1 Motivation

1.1.1 Link Scheduling: Performance Tradeoffs for Real-Time Traffic Streams

As has been noted above, real-time traffic that is unable to meet a specified timing constraint is considered lost, regardless of whether it is eventually received at

a destination node. Thus, an end-to-end deadline can be specified along with tolerable delay loss as a QOS metric for many types of multimedia traffic.

One approach to meeting end-to-end delay requirements is to *distribute* the end-to-end delay requirement across the various components of the communication path by means of an allocation policy [5]. Hence, the end-to-end or global deadline may be translated into *local* deadlines at the CPU or at intermediate network nodes as shown in Figure 1.2. By ensuring that local performance criteria are met, it is possible to guarantee that end-to-end delays are met for traffic delivered to the receiver. In the absence of local deadlines, this guarantee is hard to provide because of the variable nature of network delays.

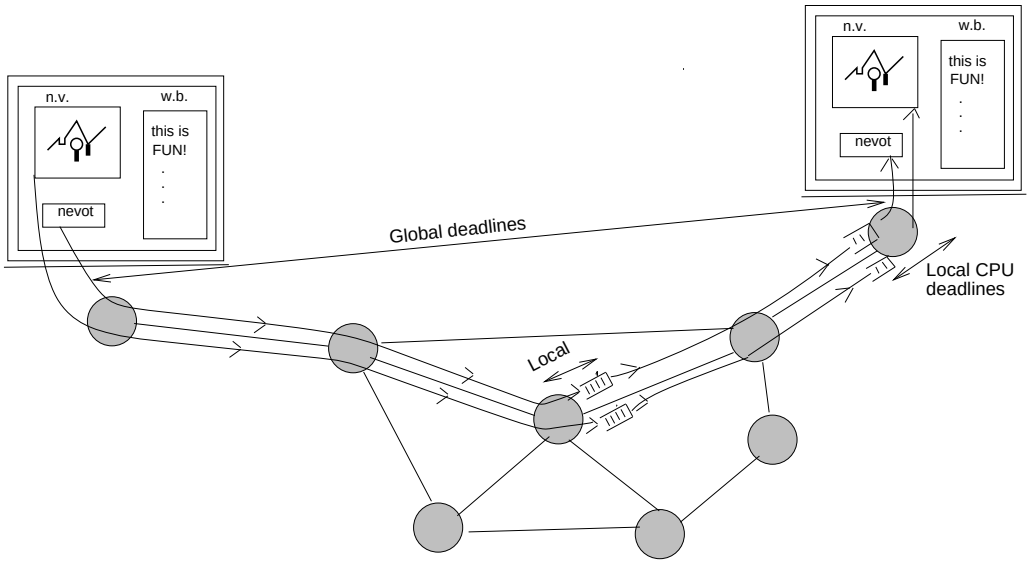

Figure 1.2. Global and local deadlines

The danger with allocating the global deadline across intermediate nodes is that a particular packet that was unable to meet a local deadline may still have been able to meet the end-to-end delay constraint. This packet would nonetheless be dropped within the network. On the other hand, if no local performance criteria are enforced, a packet that would never make the end-to-end deadline may still continue to consume

network resources unnecessarily. Thus, the particular allocation policy used must take both possibilities into account.

Local performance can also be used to determine how much additional traffic can be sustained on an end-to-end basis. Ferrandiz and Lazar [6] show how an admission control policy for new real-time sessions can be formulated based on the nodal performance of existing sessions. In general, an admission policy determines whether there exist sufficient resources to fulfill an application's request for service at the requested QOS level while continuing to observe the QOS requirement of already accepted connections. Local performance thus becomes a measure that the admission policy can use to determine the feasibility of providing required QOS to an application.

Since local performance can be of such import, due attention needs to be paid to scheduling policies at switching nodes within the network. Heterogenous real-time traffic streams can arrive at a network node. These streams may have differing local deadlines depending on the tolerable end-to-end delays and the policy used to apportion these end-to-end delays among intermediate nodes. This is shown in Figure 1.3 where $r_1, r_2, \ldots, r_k$ are local deadlines for the k different classes of traffic that leave a local node on the same output link. It then becomes the responsibility of the output multiplexer of the node to appropriately schedule packets belonging to these different streams, giving due consideration to their relative time constraints and also their relative importance. The manner in which this is done is defined by the scheduling discipline used at the output multiplexer. An important measure of the performance of a scheduling discipline is the local delay loss at the multiplexer. A good algorithm will minimize this loss.

The problem of choosing an appropriate scheduling discipline becomes even more complex when the packets of the different classes of traffic have *identical* time constraints as in the example shown in Figure 1.2. For synchronized media presentation, voice and video have to satisfy identical deadlines. However, greater clarity and

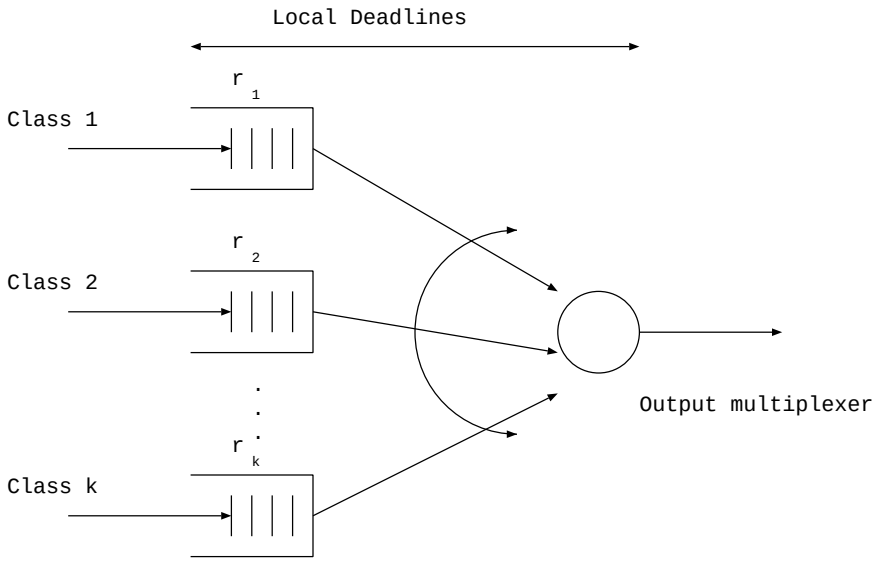

Figure 1.3. Scheduling at an output node

precision is needed in speech than in image for there to be effective communication in some applications such as conferencing. Thus, the scheduling discipline chosen should reflect this difference in performance criteria. Another example scenario could be *hierarchical source coding* for video applications wherein the digitized video signal is separated into subsignals of differing importance [7, 8]; the relatively stable background information in a picture is separated from the information pertaining to motion. The information content of a scene, in terms of bits needed to represent it, would depend on the degree of activity in the scene. While the volume of information content could be different for these different classes of traffic, all the information pertaining to a single frame would have to be available at the same time at the receiver. Thus, the deadlines for the two classes of traffic are identical (k = 2 in Figure 1.3 and $r_1 = r_2$). However, the stable background information is more important in sustaining an acceptable QOS for the users than the motion information as it can be used to refresh the screen at the receiver. Hence, once again the scheduling discipline chosen should have the ability to accord preferential treatment to the background information.

In this dissertation we consider the fundamental issue of trading the performance of two different classes of traffic with specified local deadlines at a single multiplexer by studying three different scheduling disciplines. The first discipline is priority scheduling which gives strict priority to one class of traffic over the other. The second discipline is the minimum laxity thresholding (MLT) scheme. The *laxity* of a real-time packet is the time until the expiry of its deadline. MLT is a threshold-based scheme wherein priority is given to one class of traffic when the minimum laxity of its queued packets falls below some threshold. The third scheduling discipline we study is a new "balancing" scheme that we define which assigns priorities on the basis of the differences in the minimum laxities in the two classes of traffic.

Our modeling approach and analytical methodology takes the discrete time nature of Asynchronous Transfer Mode (ATM) networks into account and builds on the work presented in [9]. In [9], a mixture of real-time and non-real-time traffic is studied. Four scheduling policies are treated there: 1) First Come First Served, 2) Priority Scheduling that always gives priority to real-time traffic, 3) a Minimum Laxity Threshold policy that gives priority to real-time traffic if the minimum laxity of the real-time traffic is less than a threshold and 4) a Queue Length Threshold policy that gives priority to non-real-time packets if their queue length exceeds a certain threshold. The performance metric of interest in [9] for real-time traffic is the probability of delay loss while for non-real-time traffic, it is average delay. In our problem, *both* classes of traffic are real-time and under the assumption of identical deadlines, we study the delay loss behavior for both classes of traffic.

We demonstrate that the new balancing scheme provides us with a parameter that we can vary to achieve better delay loss performance for *both* classes of traffic than is obtained under the other two scheduling algorithms for certain traffic arrival processes in some operating regions. We also show through simulations how, for other arrival processes, assigning strict priority to one or the other of the two classes of traffic is a reasonable choice.

1.1.2 CPU Scheduling to Minimize Preemption Costs

The problem of CPU scheduling for real-time tasks is an intriguing one and several different factors must be weighed before arriving at a suitable discipline. An example scenario for this scheduling problem can be seen in Figure 1.2. Voice and video packets could be coming off the network at periodic or near-periodic rates if the originating source generates data periodically and mechanisms such as stop-and-go queueing [10, 11] are used to preserve this traffic pattern through the network. The receiving processor would have to perform protocol processing before presenting the data to the user at a multimedia workstation. The CPU scheduling discipline has to ensure that the necessary processing is completed within a local deadline that is obtained as a portion of an acceptable end-to-end delay by an allocation policy.

The most frequently studied CPU scheduling algorithms are deadline-based dynamic algorithms and rate-based static algorithms. Of these, the earliest deadline first preemptive scheduling policy (ED) and the rate monotonic preemptive scheduling policy (RM) are common [12, 13]. Under ED, the task that has the closest deadline is always served from among all the packets that are awaiting service at the host. This is equivalent to serving the job with the minimum laxity. Under RM, the earliest arriving packet of the type with the highest frequency is served first. Thus, under this policy, fixed priority assignments are made and deadlines are not considered explicitly in making scheduling decisions.

Both ED and RM are *preemptive* policies. Thus, under ED, for example, if a job which has to complete service prior to a given time is being served when another job that has to finish service by an earlier time arrives, the CPU stops serving the job currently being served and starts serving the arriving job instead. The same is true under RM if the arriving job is of higher frequency than the one currently under service. There is a cost associated with preemptions called the preemption overhead time during which no effective work associated with any of the jobs in the system can be done by the CPU. In this dissertation, we study preemptive scheduling algorithms

by looking at the potential preemption overhead that an algorithm introduces. Recent studies [14, 15] have demonstrated that this overhead merits serious consideration in actual workstation implementations. We show here that the number of preemptions under ED is always smaller than the number of preemptions under RM. This shows that the preemption overhead under ED is no worse than that under RM and– as our simulation study indicates– may, on occasion, be markedly lower. For example, our simulation results show that the difference between the number of preemptions under each discipline can exceed 20% for the workloads [16] considered. These simulation results in combination with the well-known fact [12] that greater processor utilizations can be supported under ED than under RM enable us to present arguments in support of using ED as the algorithm of choice for workstation processor scheduling.

1.1.3 Scalability in Multicast Error Recovery Protocols

We have recently seen the widespread introduction of applications that support real-time interactive group collaborations over wide area networks (WANs). These include applications that support video (nv [3]) and voice (vat [17], NeVot [2]), which do not require reliable multicast, as well as applications such as shared whiteboards (e.g., wb [4], shdr [18]) which do require reliable multicast. This requirement of reliable data transfer for this last set of applications poses a difficult challenge to network protocol designers– namely how to design and implement a reliable multicast protocol that can handle hundreds or thousands of participants[1]. In such an environment, traditional error recovery schemes based on positive acknowledgments (ACKs) and timeouts at the sender, will lead to what has come to be described as the *ACK-implosion* problem [20, 21]. On receiving an ACK, the sender would have to reset a timer and perform some amount of processing to keep track of which receiver has acknowledged which piece of data. A single sender that communicates with multiple receivers will quickly be overwhelmed by the ACK processing overhead

[1] A recent IETF conference was multicast to several hundred workstations world-wide [19].

imposed by hundreds of receivers (see Figure 1.4). Hence, attention needs to be paid to the *host processing* requirements of any protocol for reliable data transfer in such environments.

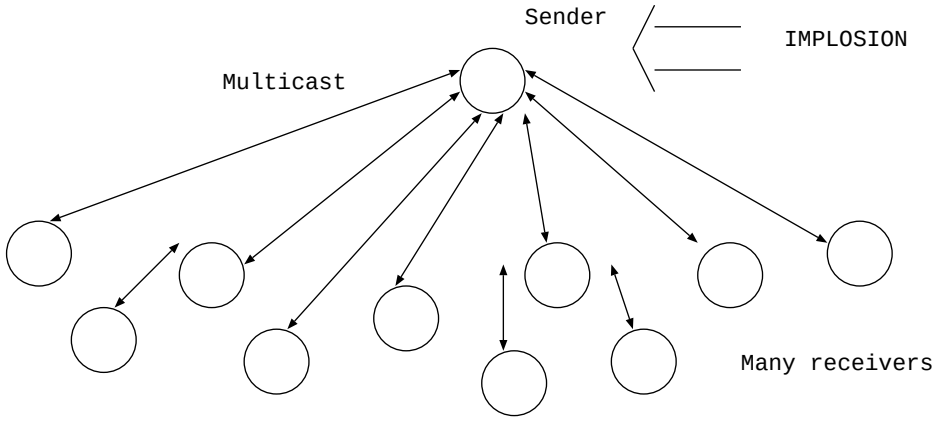

Figure 1.4. ACK implosion at sender

Broadly, there are two different approaches to providing reliable, scalable multicast communication. The first is the traditional *sender-initiated* approach which places the responsibility for reliable data delivery on the sender. In particular, the sender is required to maintain state information regarding all receivers to which it is multicasting data. Reliability and maintenance of this state information is ensured by having receivers return positive acknowledgments for every packet correctly received, and the use of timers at the sender for the purpose of detecting packet losses. The alternate approach, the *receiver-initiated* approach, shifts most of the responsibility for reliable data delivery to the receivers. Each receiver is responsible for detecting lost packets and informing the sender via negative acknowledgments (NAKs) when it requires the retransmission of a packet.

In this dissertation we study three generic protocols to ensure reliable and scalable multicast, one that is sender-initiated and two that are receiver-initiated. We focus on the *fundamental differences* between them, i.e., the use of ACKs versus NAKs, the use of point-to-point channels for NAKs versus broadcasting NAKs. Further,

recognizing that communication bandwidths are expected to grow at a much higher rate than processing speeds during the next decade, we focus on the *processing requirements* of these protocols at both the sending and receiving hosts rather than on the communication bandwidth requirements. Previous analyses have focussed on the bandwidth requirements of different reliable multicast protocols. We are also primarily concerned with *scalability*: how well each of these approaches will handle large numbers of receivers.

We observe through simple analyses that a simple receiver-initiated protocol which requires receivers to return negative acknowledgments (NAKs) to the sender over point-to-point channels provides substantially better performance (in terms of the maximum supportable throughput of successfully transmitted messages) than a sender-initiated protocol. Further, substantial improvement is obtained by the multicasting of NAKs coupled with the introduction of random delays prior to the transmission of a NAK.

1.2 Contributions of this Dissertation

The following are the contributions of this work:

- We define a *new* balancing scheduling algorithm, and compare its delay loss performance to those of two other well-known scheduling disciplines (MLT and priority scheduling) for two different classes of traffic that have some given time constraints. There has been previous work in the area of scheduling to meet deadline constraints for many classes of real-time traffic (e.g., [22]) and to meet quality-of-service requirements on mixtures of real-time and non-real-time traffic (e.g., [9]) but very little has been done on trading the performance of one class of real-time traffic against that of another. The new balancing scheduling algorithm that we define has a parameter that can be varied to achieve tradeoffs in the delay loss performance of the two classes of traffic. We show that we are able to achieve superior performance for some arrival processes using the new scheme

as compared to priority scheduling and MLT. For example, for some operating regions we are able to achieve at 29% drop in the loss probability of Class 2 (when compared to strict priority) with negligible change in the loss probability of Class 1 traffic while using the new scheme for the traffic arrival processes considered. We are thus able to argue that our scheme is to be preferred to the traditional schemes of MLT or priority scheduling.

- We compare the preemption performance of two well-known CPU scheduling algorithms– ED and RM– and prove analytically for a general workload that ED, which is a dynamic priority assignment, always has fewer preemptions than the static priority assignment (RM). We show by simulation that the number of preemptions can be as much as 20% greater for a standard fixed priority scheme like RM as compared to ED. For other fixed priority schemes, the difference in the number of preemptions relative to ED can be over 188%. It has traditionally been argued that the static priority schemes are easier to implement. A closer look at the actual implementation details suggests that this argument is erroneous and that there is only marginal difference in the implementation complexity of the ED and RM algorithms. Thus, our results show that ED is to be preferred for processor scheduling.

- We demonstrate through analysis and numerical results on different workloads that receiver-initated, NAK-based protocols provide better scalability performance than sender-initiated, ACK-based protocols for reliable and scalable multicast when host processing capacity is the constrained resource. Much of the recent work on multicast applications has been focussed on network issues such as routing [23]. Although there have been attempts at defining protocols that handle sender issues such as the ACK-implosion problem [21], there has been no work on explicitly modeling *host* protocol processing overheads. We perform host throughput analyses for one sender-initiated and two receiver-initiated pro-

tocols. We show using complexity expressions and numerical results for specific workloads that there can be orders of magnitude difference in the performance of the different protocols studied.

1.3 Structure of this Dissertation

In this introduction, we have provided the context and motivation for three problems in the areas of scheduling and error recovery protocols for multimedia applications.

The rest of this dissertation is organized as follows. In the next three chapters we provide additional technical details for each of the three problems. In each chapter we present a survey of other work in the specific areas that we consider.

In Chapter 2 we discuss the problem of link-level scheduling for two classes of real-time traffic. We describe the characteristics of the system that we study. We then discuss the model we use and our analysis methodology. We present analytic results for the performance of various scheduling disciplines for a particular geometric bulk arrrival traffic model. We also describe a superposed voice traffic model and present simulation results for this model.

In Chapter 3, we discuss the preemption performance of different CPU scheduling algorithms. We provide the proof of our claim that ED has fewer preemptions than RM and show corroborating simulation results. We also provide simulation results comparing numbers of preemptions for different scheduling disciplines in a complex system in which various operating system costs are modeled.

In Chapter 4, we discuss the scalability of different error-control protocols that can be used in multicast applications. We provide the throughput analyses for three such generic protocols and provide complexity expressions to demonstrate how they scale to large numbers of receivers. We compare the protocols numerically both when different components of host processing have equal costs and when they have unequal costs.

In Chapter 5, we summarize the dissertation and indicate future research directions.

CHAPTER 2

PERFORMANCE OF LINK SCHEDULING ALGORITHMS

2.1 Introduction

In this chapter we consider the issue of link-level scheduling of two classes of real-time traffic and examine the tradeoffs possible in their delay loss performance under different scheduling disciplines. As has been noted in Chapter 1, this is an interesting problem in multimedia applications in which different classes of traffic with end-to-end timing constraints have different performance criteria to meet. These end-to-end timing constraints can be translated into local (or nodal) deadlines by an allocation policy. Scheduling policies can then implemented at switching nodes in the network. Possible applications of interest include those that make combined use of voice and video which have to presented at the destination in a synchronized way, or hierarchical source coding for video where the source traffic is separated into subbands of differing importance.

We begin this chapter by surveying related work in the literature. We then describe the characteristics of the system that we study and define the specific link-level scheduling problem that we address. We present the model that we use for our analytic study, define the three scheduling policies that we study: strict priority, minimum laxity thresholding and the balancing discipline. We analytically solve for loss probabilities, and show how the new balancing scheduling discipline that we define provides better loss performance than the other two policies for both classes of traffic for certain arrival processes. We then describe a superposed voice traffic model for our simulation study and show that with this arrival process, assigning strict priority to one or the other class becomes the preferred option.

2.2 Survey of Related Work

For effective interactive communication between users of multimedia applications, it is necessary to limit the end-to-end delays. For example, pauses greater than 100 milliseconds become noticeable to the human ear. Achieving these bounded delays for networked applications is a challenging problem. One way of achieving these bounded end-to-end delays is to impose local delay bounds at intermediate nodes in the network. There has been considerable research effort directed at the problem of studying packet scheduling mechanisms for real-time traffic at intermediate nodes and also at designing end-to-end mechanisms for bounding delays. Here we look at some of this work.

Panwar et. al. [22] consider the problem of scheduling impatient customers at a single server queue. Each customer has a deadline drawn from a general independent distribution. The Shortest Time to Extinction (STE) scheduling discipline is studied here. STE is very similar to another scheduling algorithm called Earliest Due Date (EDD). Under EDD, customers are served according to non-decreasing due dates and STE differs from this in that a customer whose due date is already past is never scheduled for service. It is shown that STE is optimal for single server queues in the sense of minimizing delay loss. The deadlines here are measured from arrival time to the beginning of service and the metric that is maximized is the fraction of customers that begin service within their respective deadlines. The issue of trading off losses for different classes of traffic is not treated here.

Lim and Kobza [24] consider the problem of scheduling multiple classes of delay sensitive traffic at a network node. They propose and analyze a link-level priority scheduling discipline called Head-of-the-Line with Priority Jumps ($HOL - PJ$). In this discipline, each class of traffic is assigned a distinct priority level and the packet that is at the head of the highest priority queue receives service. However, if a packet has not been served by a delay limit at the current queue, it jumps to the tail of the next higher priority queue. Through the suitable adjustment of the delay limits of

each queue, this scheme can be used to to meet average delay criteria for each class of traffic. While this scheme does impose a certain delay limit on each of the queues except for the highest priority one, packets that have failed to meet their end-to-end delay requirement in a network setting will still receive service at this node. This violates the notion of "better never than late" for real-time traffic, i.e., packets that are already too late to be useful at the destination should not be permitted to use up network resources. Schulzrinne et. al. [25] explicitly model local deadlines and drop real-time packets that will be unable to meet them. The authors propose the discarding of packets based on the amount of virtual work that an arriving packet sees at a node as a congestion control mechanism and study the overall loss performance of the traffic– both due to delays and due to discarding.

Verma et. al. [26] study the feasibility of bounding the delay *jitter*, i.e., the variation in end-to-end packet delays, for real-time channels. This is based on earlier work [27] in which the goal is to provide end-to-end delay bounds by creating local delay bounds at intermediate nodes at the time of channel establishment. Packet delay bounds are either expressed deterministically as absolute values, or statistically as a requirement that the probability of meeting a certain delay bound be greater than a given value. At each intermediate node, three queues are maintained: one for deterministic packets, the second for statistical packets and the third for all other types of packets such as those that are non-real-time. The scheduling policy used is deadline-based and is a multiclass version of the Earliest Due Date discipline. The scheduler picks a deterministic packet if the time by which it has to begin service is before the time by which a statistical packet would end service. Real-time traffic is accorded higher priority than non-real-time traffic. There is rate-based flow control on the accepted traffic, i.e., an upper limit is imposed on input rate to the channel. These mechanisms serve to establish a bounded-delay server. In [26], the bounded-delay server of [27] is used and the added requirement is imposed that each node preserve the timing pattern of arriving packets at the input end of the channel. This provides

bounds on the delay jitter. However, the authors do not consider the case of several classes of traffic between the same source-destination pair. Therefore, the issue of trading off the performances of different classes of traffic does not arise in this work.

Clark et. al. [28] argue that some applications (such as voice conversations) are able to tolerate a proportion of the packets missing the end-to-end delay bound. Hence, they introduce scheduling for *predicted* service in which the measured performance of the network is used in computing delay bounds. In [28] the authors propose an algorithm called $FIFO+$. The average delay for each aggregate class of traffic is computed for each hop and for each packet the difference between its particular delay and the average delay is added to a field in its header. Output multiplexers serve packets according to their expected arrival times rather than actual arrival times. $FIFO+$ is able to slightly decrease the mean delay and considerably decrease the jitter as compared to $FIFO$ for packets traversing many hops along the source-destination route. More recently, Schulzrinne et. al. [29] compared the performance of another algorithm called hop-laxity (HL) scheduling to that of $FIFO+$. Under HL, the ratio of the laxity to the number of remaining hops to destination is computed for each packet and the packet with the least such ratio is served first. The delay characteristics of HL are similar to those of $FIFO+$.

Our work differs from all of the above in that we explicitly trade the performance of one class of traffic against that of another, and examine whether scheduling at network nodes is an effective means of achieving adequate performance for all classes of traffic. We study the performance of packet-scheduling algorithms when different classes of traffic have identical nodal time constraints in detail.

2.3 System Description

As noted earlier, the real-time applications that provide the motivation for this work operate in a widely distributed networks. It is difficult to provide tight bounds on end-to-end delays in such environments. For instance, round-trip delays in the

Internet for users separated across the continental United States can be in the range of hundreds of milliseconds, with large variations based on the route taken, network characteristics for the day and time at which the application is run etc.

One way of ensuring bounds on the end-to-end delays is by imposing limits on the number of hops that a packet may traverse and on the permissible local delay at each intermediate hop. NeVot [2] is an audio tool that can be used to communicate over the Internet and can be used as an example to clarify this point. We can assume a maximum hop count of 15 between a pair of communicating nodes for a permissible one-way delay of 120 milliseconds plus propagation delays. In order to meet this end-to-end delay, we could impose a maximum delay of 8 milliseconds at each node. NeVot uses 160 byte packets plus headers. The transmission time of such a packet on a T1 link of 1.544 Mbps is 0.83 milliseconds. Thus, an arriving packet has a local deadline of 10 time slots where a time slot is normalized to the time to transmit a packet. For higher link speeds (such as the T3 link), permissible local deadlines will be correspondingly higher. These figures give us an idea of the range of likely local deadlines for real applications.

In general, different streams (or classes) of data can arrive at an intermediate switching node in the network and depending on their destinations, all or some of these streams may compete for the same outgoing link at the node. Here we consider a multiplexer at the output buffer of the switching node which must decide which packet from which stream is to transmitted next on a given outgoing link (see Figure 2.1). The different types of data could all potentially have different local deadlines and the scheduling policy used must take these values into account in addition to the relative importance accorded to the various classes of traffic. We specifically consider two classes of traffic bound for the same destination and examine the performance of the multiplexer under different scheduling discipines.

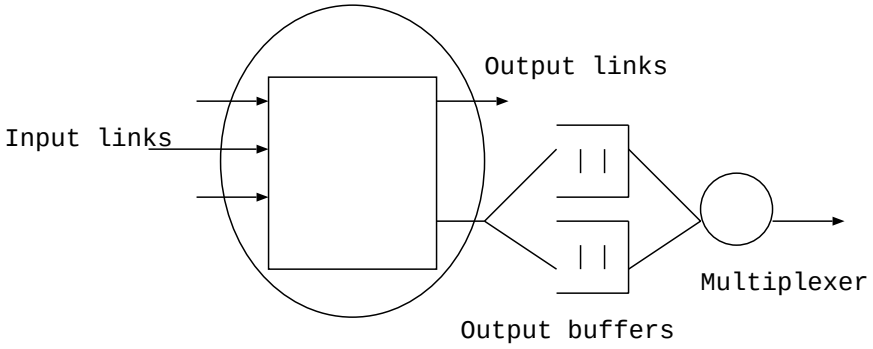

Figure 2.1. Switching node in the network

2.4 Metrics and Policies

Any real-time packet that does not reach its destination within the specified deadline is lost and thus the delay loss probability captures the performance of a particular policy for a particular class of real-time traffic. The delay loss probabilities of the two classes of traffic may be traded off against one another by various scheduling policies. We consider only delay losses and assume that there is adequate buffer space at the multiplexer, i.e., there are no buffer overflow losses. Three scheduling policies are studied here.

2.4.1 Priority Discipline

In this scheduling policy, priority is always given to Class 1 traffic. Class 2 traffic is transmitted (served) only if there are no queued Class 1 packets. Within a class, packets are served FCFS.

2.4.2 Minimum Laxity Thresholding (MLT)

The laxity of a real-time packet is the time until the expiry of that packet's deadline. When a packet first arrives at a queue at the scheduler, its laxity is equal to its deadline and with each passing time slot, its laxity decreases by one. In the MLT discipline [9], a threshold is specified on the laxity of Class 1 traffic. If the

minimum laxity of the queued Class 1 packets is less than or equal to the threshold, or there are no queued Class 2 packets, the minimum laxity Class 1 packet is served. The queued minimum laxity Class 2 packet is served either if the laxity of minimum laxity Class 1 packet is greater than the threshold or if there are no queued Class 1 packets. When the threshold T is equal to the deadline of Class 1 traffic, MLT reduces to the priority discipline. Reducing T increases the relative importance accorded to Class 2 traffic.

Let x_1 and x_2 be minimum laxities of queued Class 1 and Class 2 packets. Further, let $x_1 = 0$ and $x_2 = 0$ correspond to the case of there being no Class 1 and no Class 2 packets in the system respectively. We can write the MLT policy formally as shown below:

if ($(x_1 \neq 0)$ .and. $((x_1 \leq T)$ .or. $(x_2 = 0))$)

serve minimum laxity Class 1 packet;

else if $(x_2 \neq 0)$

serve minimum laxity Class 2 packet.

2.4.3 Balancing Discipline

In this scheduling discipline, a quantity B is specified with reference to the difference between the laxities of the minimum laxity Class 1 and Class 2 packets. A Class 1 packet is served *unless* the laxity of the minimum laxity Class 2 packet is at least B smaller than the laxity of the minimum laxity Class 1 packet. Thus, B becomes a parameter that can be varied to change the relative priorities of the two classes. When B is equal to the deadline of Class 1 traffic, the balancing discipline reduces to the priority discipline.

Using the same notation as in the MLT case, we may write the balancing discipline more formally as follows:

if ($(x_2 \neq 0)$ .and. $((x_1 - x_2) \geq B)$)

serve minimum laxity Class 2 packet;

else if $(x_1 \neq 0)$

serve minimum laxity Class 1 packet.

As can be seen from the above descriptions, the MLT and balancing schemes provide us with parameters T and B, respectively, which can be varied to effect tradeoffs between the loss of Class 1 and Class 2 traffic. Both MLT and the balancing scheme become the same as the priority discipline in limiting cases.

2.5 Modeling and Analysis

2.5.1 Assumptions for Analysis

We consider a discrete-time multiplexer at the output buffer of a switching node that makes the scheduling decision with regard to two arriving classes of real-time traffic with local deadlines r_1 and r_2. The two classes of traffic can be thought of as being queued separately and the multiplexer selects a single fixed length packet from either queue for transmission during a slot.

We assume that the arrival streams of the two classes of traffic are independent of each other. We will initially assume that arrivals in a slot are independent of arrivals in all other slots. For each class of traffic, arrivals in a slot are treated as bulks with the bulk sizes being geometrically distributed. All arrivals in a slot are assumed to have occurred just prior to the beginning of the next slot. Each arriving packet of class k also has an associated laxity equal to the prespecified deadline, r_k. A packet which is not transmitted within its deadline is considered lost, and is removed from the queue. All arrivals in the same slot of a particular class have the same chance of being chosen for transmission in succeeding slots. Figure 2.2 illustrates the scenario we consider.

2.5.2 Solution Approach

The system is modeled as a two dimensional Markov chain (x_1, x_2) where x_k is the laxity of the minimum laxity class k packet (nominally at the head of the queue

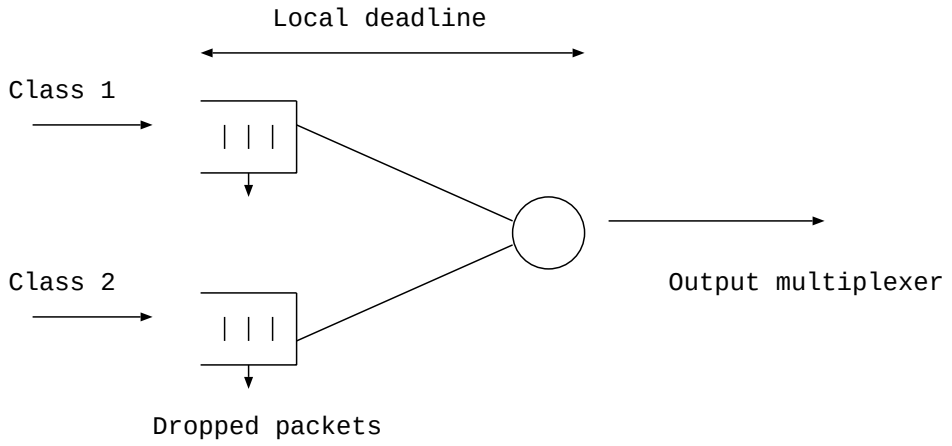

Figure 2.2. Scheduling two classes of real-time traffic

of class k packets). This model is possible because of the assumption of geometrically distributed bulk sizes with independence from slot to slot, which enables us to write state transition probabilities in a Markovian manner. The possible values for x_1 are $0, 1, 2, 3, \ldots, r_1$ and for x_2 they are $0, 1, 2, 3, \ldots, r_2$ where 0 represents the case of there being *no* packets of the corresponding class of traffic. Hence, there are $(r_1+1)\times(r_2+1)$ possible states of the system.

There exist clearly definable transition probabilities from one state to another. Due to the Markovian nature of the model, these transition probabilities depend only on the current state of the system– i.e., the probability of reaching a particular state in the following time slot depends only on the current state and the particular scheduling discipline used. Thus, we can write a matrix of state transition probabilities for the discrete time Markov chain. This matrix will be of dimension $((r_1 + 1)(r_2 + 1)) \times ((r_1 + 1)(r_2 + 1))$. The entries in this matrix will depend on the scheduling discipline used and represent transitions from state (x_1', x_2') to state (x_1'', x_2'').

Using standard techniques, the transition matrix can be used to obtain the state occupancy probabilities in steady state. Once these state probabilities have been obtained, the throughput of each class of traffic can be obtained by simply summing probabilities over those states in which the scheduling discipline will choose a packet

from that particular class of traffic for transmisson in the next slot. For example, for the priority discipline, if the system is in state $(4,2)$, Class 1 traffic is transmitted in the next slot and thus this state contributes to throughput of Class 1 traffic. For the same discipline, if the state is $(0,2)$, Class 2 traffic is transmitted in the next slot. More generally, if $\Pi_{i,j}$ gives the state occupancy probability for system state (i,j) and γ_k is throughput of class k, we can write throughput expressions for the priority case as follows:

$$\gamma_1^{pri} = \sum_{i=1}^{r_1} \sum_{j=0}^{r_2} \Pi_{i,j} \tag{2.1}$$

and

$$\gamma_2^{pri} = \sum_{j=1}^{r_2} \Pi_{0,j}. \tag{2.2}$$

Throughput equations for the MLT case are given by

$$\gamma_1^{mlt} = \sum_{i=1}^{T} \sum_{j=0}^{r_2} \Pi_{i,j} \tag{2.3}$$

and

$$\gamma_2^{mlt} = \sum_{i=T+1}^{r_1} \sum_{j=1}^{r_2} \Pi_{i,j} + \sum_{j=1}^{r_2} \Pi_{0,j} \tag{2.4}$$

Throughput equations for the balancing case are

$$\gamma_1^{bal} = \sum_{i=1}^{r_1} \sum_{j=i-B+1}^{r_2} \Pi_{i,j} + \sum_{i=1}^{r_1} \Pi_{i,0} \tag{2.5}$$

$$\gamma_2^{bal} = \sum_{i=2}^{r_1} \sum_{j=1}^{i-B} \Pi_{i,j} + \sum_{j=1}^{r_2} \Pi_{0,j} \tag{2.6}$$

For all three disciplines, if the arrival rate for Class k is λ_i, then we can write the probability of loss for Class k traffic, $Ploss_k$ as

$$Ploss_k = \frac{\lambda_k - \gamma_k}{\lambda_k} \tag{2.7}$$

Details of the actual form of the transition matrix are given in Appendix A.

2.6 Numerical Results

In order to quantitatively evaluate the performance of the three scheduling disciplines, we used the method described in Appendix A and determined the transition probability matrices for the specific situation of $r_1 = r_2 = r$. Standard numerical techniques were used to solve the matrix equations for state occupancy probabilities in steady state. This was done in each case for the specific deadline values of 10 and 30. The value of 10 is suggested by the characteristics of NeVot audio packets as discussed in Section 2.3. The value of 30 was chosen because for 53 byte packets, the transmission time on a T1 link for a single packet is 0.28 milliseconds. If the local deadline is around 8 milliseconds, a local deadline of 30 slots becomes a plausible figure. For these deadline values, throughputs for each class of traffic were obtained from the state occupancy probabilities in the manner described in the previous section. Delay loss probabilities were obtained via Equation 2.7.

Figure 2.3 illustrates the tradeoffs between $Ploss_1$ and $Ploss_2$ for $r = 10$ for the case of balanced traffic. Here the arrival rates of the two classes of traffic $\lambda_1 = \lambda_2 = 0.3$. $Ploss_2$ is plotted on the X-axis on a linear scale and $Ploss_1$ on a log scale along the Y-axis. For the MLT and balancing disciplines, the parameters T and B can respectively be varied to yield a set of achievable performance levels. For the priority discipline, there is no such parameter and there is only one point in the graph that represents this policy. The parameters T and B vary between 1 and 10 and increase with increasing X-axis values. As can be seen, both MLT and balancing tend to the priority case in the limiting cases of $T = 10$ and $B = 10$, respectively, where all three disciplines have identical performance. The lines joining the points are indicated for clarity– the points themselves are obtained through independent solutions to the set of state transition equations.

As can be seen from Figure 2.3, as T increases, $Ploss_1$ decreases in the MLT case. This happens because the likelihood of serving Class 1 increases. For the balancing case, as B increases, $Ploss_1$ decreases. This is true because as B increases, Class 2

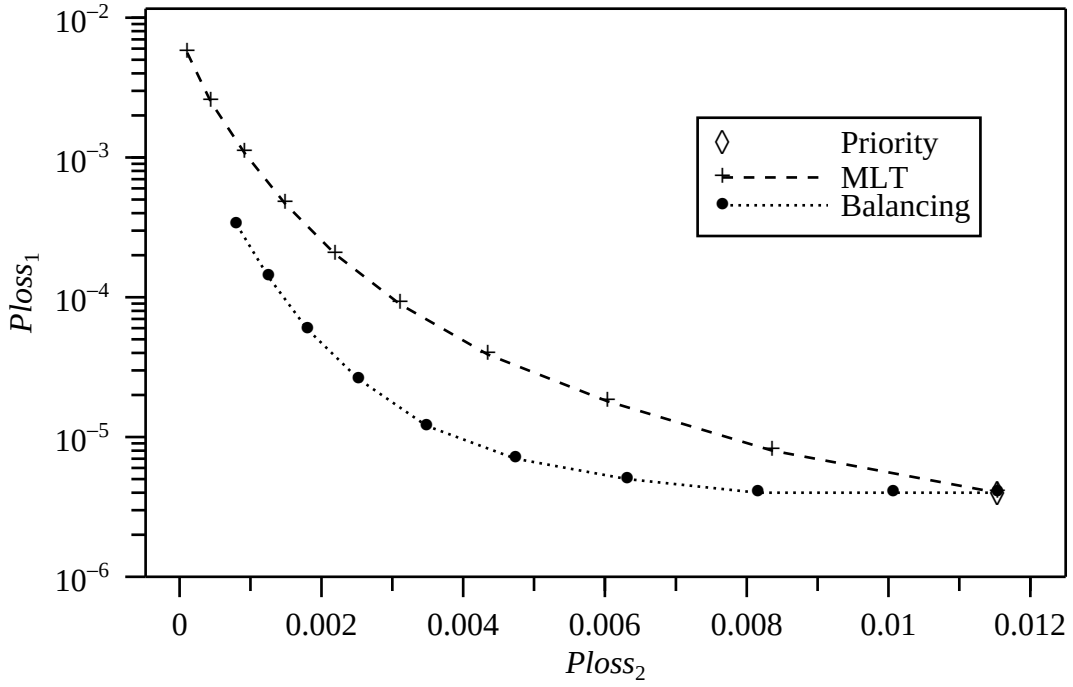

Figure 2.3. $\lambda_1 = \lambda_2 = 0.3, r = 10$

minimum laxity has to be much smaller than Class 1 minimum laxity for a Class 2 packet to be served. Consequently, the likelihood of serving Class 1 increases, thereby reducing loss probability for Class 1.

It can be seen that the balancing scheme achieves lower delay loss (for *both* classes) over a range of T and B values. By reducing B, we move towards the left along the balancing curve and increase the relative importance given to Class 2 traffic. Looking at the flat portion of the balancing graph in Figure 2.3 shows us that a useful tradeoff can be obtained between $Ploss_1$ and $Ploss_2$. For example, a halving of $Ploss_2$ is possible in this region while still keeping $Ploss_1$ at a low level: $Ploss_2$ drops from 0.012 to 0.006 while $Ploss_1$ is kept below 10^{-4}.

While the results examined thus far favor the balancing scheme, it must be pointed out that the MLT case can always attain a lower minimum value of $Ploss_2$ than the one achievable by the balancing case. Further, while the gains in Class 2 loss performance are impressive under the balancing scheme, they are not in the orders

of magnitude. To probe these aspects further, we plotted both $Ploss_1$ and $Ploss_2$ for $\lambda_1 = \lambda_2 = 0.3$ along log scales in Figure 2.4. In this figure we also plotted the data obtained when the positions of the two classes of traffic are switched from a scheduling perspective. We obtained these additional data points by imposing thresholds on minimum laxity Class 2 packets instead of on Class 1 packets for the MLT policy, and by using negative values of B for the balancing policy. These data points are the mirror images of those obtained earlier.

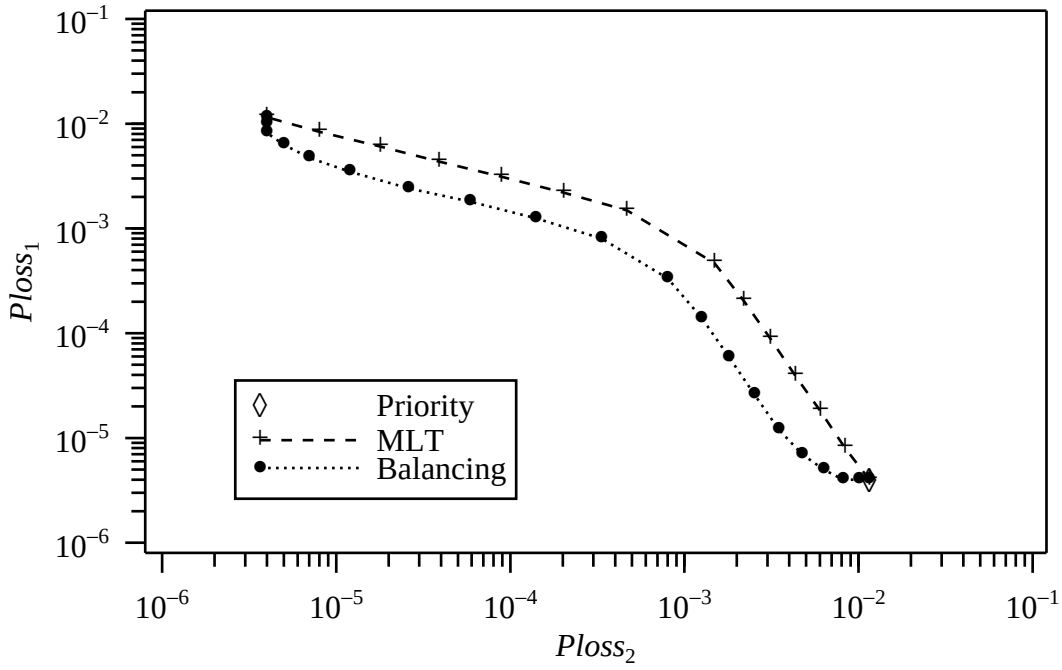

Figure 2.4. Extended T and B, $\lambda_1 = \lambda_2 = 0.3, r = 10$

As can be seen from Figure 2.4, while the extreme points of MLT and balancing are the same and correspond to giving strict priority to either Class 1 or Class 2 traffic, the balancing curve lies under the MLT curve. Thus, for any given value of T, we can find a value of B for which the delay loss probabilities of both classes of traffic are lower (or at least equal in the extreme cases) under the balancing scheme than under MLT. However, one can also see in this graph that under both balancing and MLT,

that there cannot be orders of magnitude improvement in the performance of one class of traffic without paying a corresponding price for the other class of traffic. While the balancing scheduling discipline does provide relative benefits over the MLT scheme if moderate losses can be tolerated for both classes of traffic, if the application demands extremely low loss for one or the other class of traffic, strict priority scheduling is the best option.

The differences between the scheduling disciplines become even smaller as we increase the arrival rates. This can be seen in Figures 2.5 and 2.6 where the arrival rates are $\lambda_1 = \lambda_2 = 0.4$ and $\lambda_1 = \lambda_2 = 0.45$ respectively.

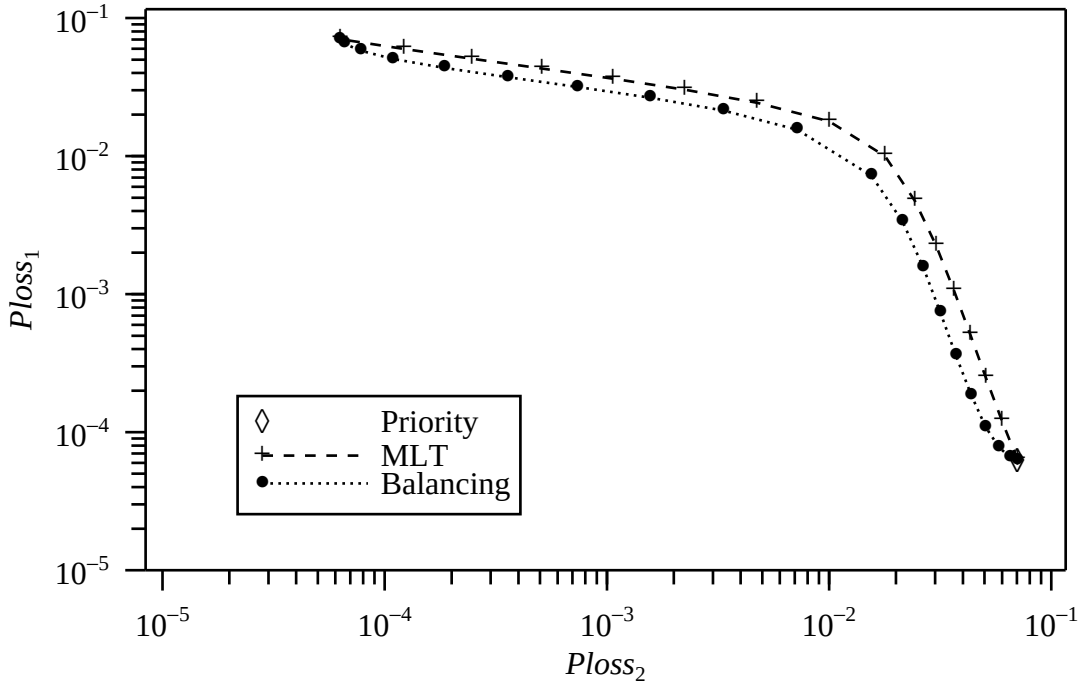

Figure 2.5. $\lambda_1 = \lambda_2 = 0.4, r = 10$

As may be expected, the overall delay loss probabilities for both classes of traffic increase as arrival rates increase and the deadline remains constant. This can be seen by simply comparing the axes values in Figures 2.4, 2.5 and 2.6. However, as noted before, the gap between MLT and balancing narrows with increasing arrival rates. Further, it becomes clear by comparing these figures that for high arrival rates, a

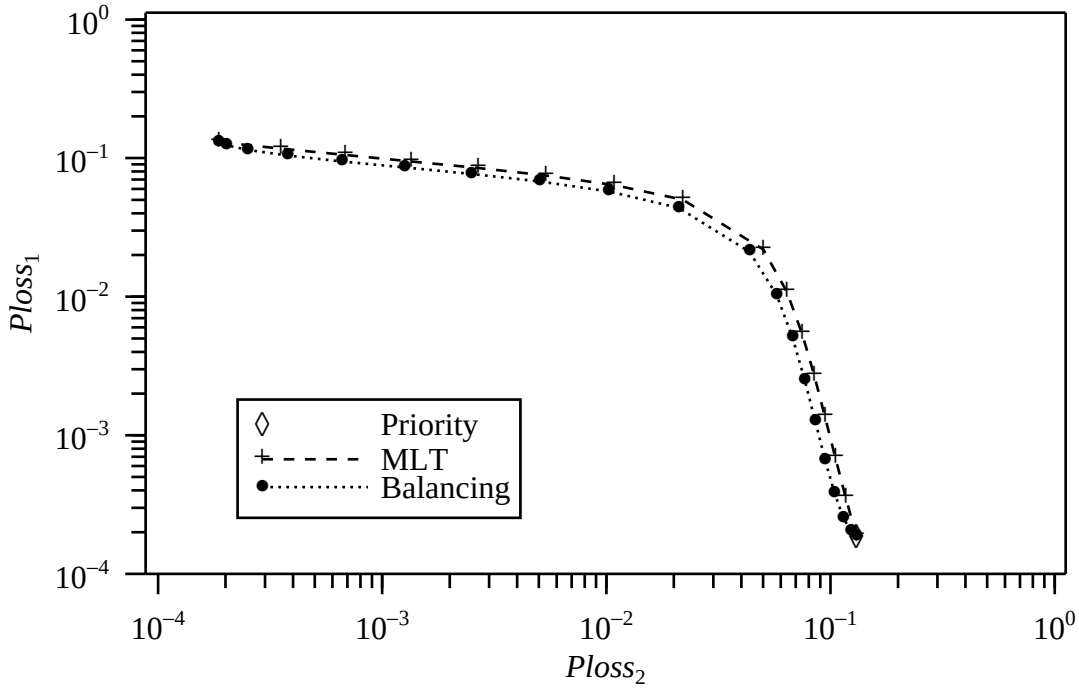

Figure 2.6. $\lambda_1 = \lambda_2 = 0.45, r = 10$

good option is to always accord strict priority to one or the other class of traffic. The steep slopes of the graphs show that varying the parameters T and B gives us very little "play" in terms of reducing the delay loss probability of one class of traffic without a steep increase in the delay loss probability of the other class of traffic.

In Figure 2.7 we keep the arrival rates at $\lambda_1 = \lambda_2 = 0.45$ but increase the deadline r to 30. The overall delay loss probabilities are lower than when the deadline is 10. For either class of traffic, the delay loss probabilities can fall on occasion to very low values that do not appear on our graphs. This is true at either extreme of according strict priority to one or the other class of traffic. Here too we see that the best option is to operate at one or the other extreme as the parameterized scheduling disciplines of MLT and balancing provide little additonal benefit. This argument is the same as for the case when $r = 10$.

We now turn our attention to the behavior of the scheduling disciplines for cases where the arrival rates of the two classes of traffic are not equal but packets of both

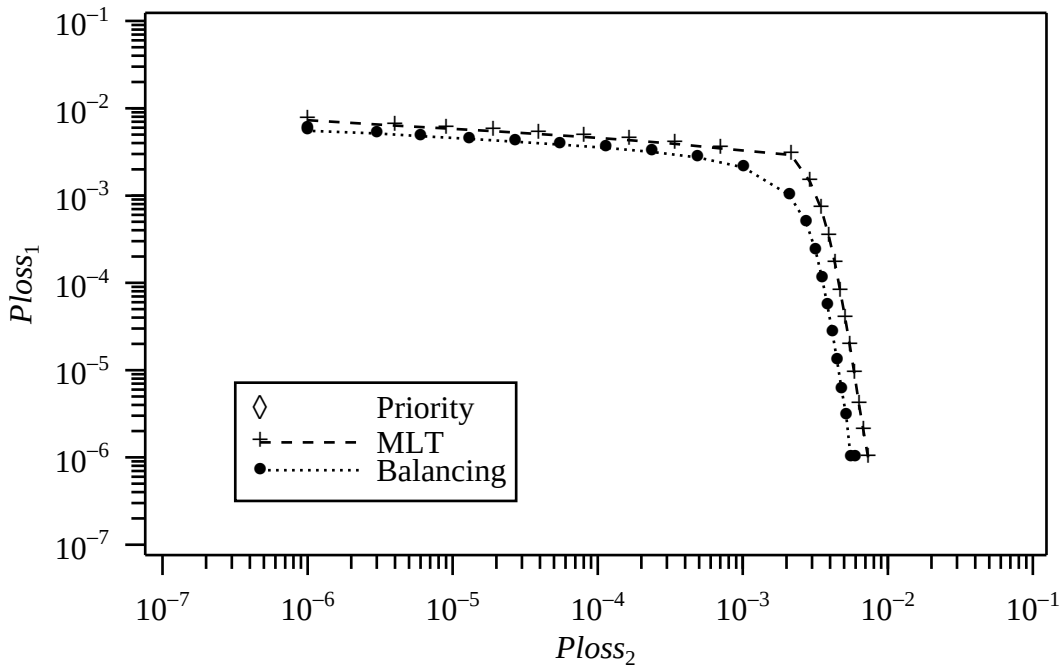

Figure 2.7. $\lambda_1 = \lambda_2 = 0.45, r = 30$

classes have a deadline of 10. Figure 2.8 shows the delay loss tradeoffs possible for $\lambda_1 = 0.15,\ \lambda_2 = 0.75$. Figure 2.9 shows the results for $\lambda_1 = 0.75,\ \lambda_2 = 0.15$. For both these figures we revert to our original schemes of varying T and B between 1 and 10. We do not plot the extended data points as in Figures 2.4 through 2.6 because here we wish to comment on the effect that low values of threshold for MLT have when the relative proportions of the two classes of traffic are changed. In both combinations of arrival rates for the two classes, the total arrival rate is 0.9.

The curves in Figure 2.8 do not extend all the way to the point at which strict priority is given to Class 1 traffic. This is because the loss probability of Class 1 traffic falls to a value below the minimum value on our log scale.

The conclusions that we can draw from from Figures 2.8 and 2.9 are interesting. In general, $Ploss_1$ values are lower for lower Class 1 arrival rates for both MLT and balancing. However, under MLT the maximum value of $Ploss_1$ is *lower* for $\lambda_1 = 0.75$ (Figure 2.9) than for $\lambda_1 = 0.15$ (Figure 2.8). This happens because for low values of

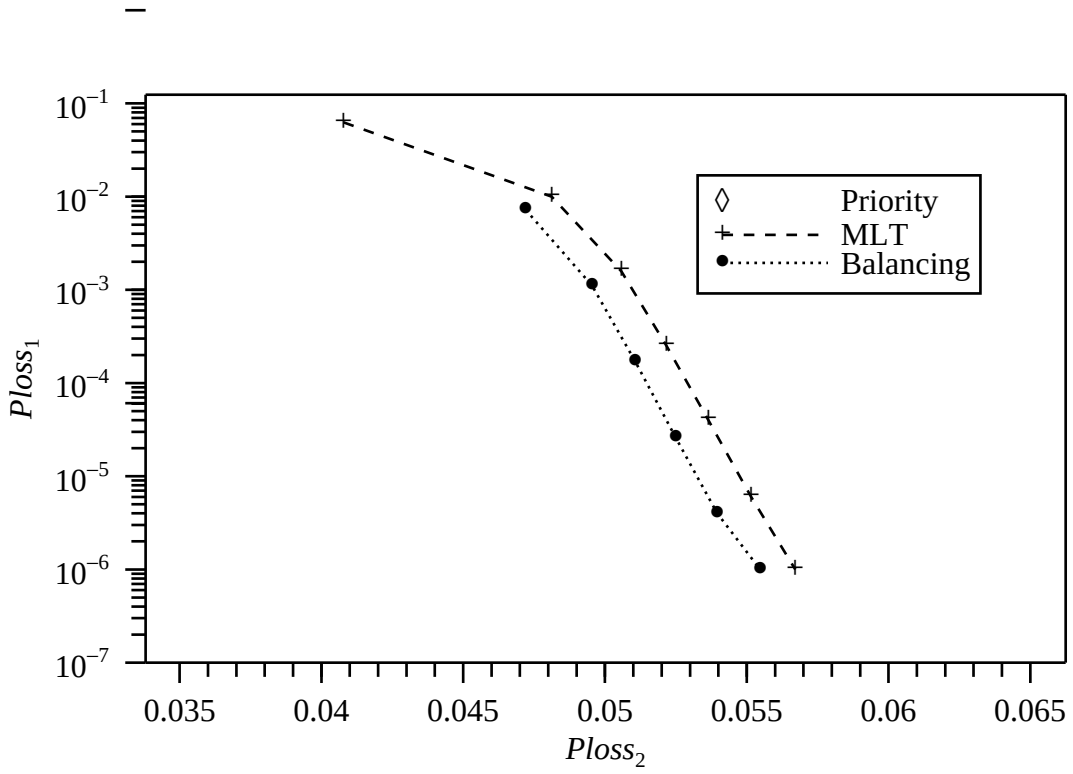

Figure 2.8. $\lambda_1 = 0.15, \lambda_2 = 0.75, r = 10$

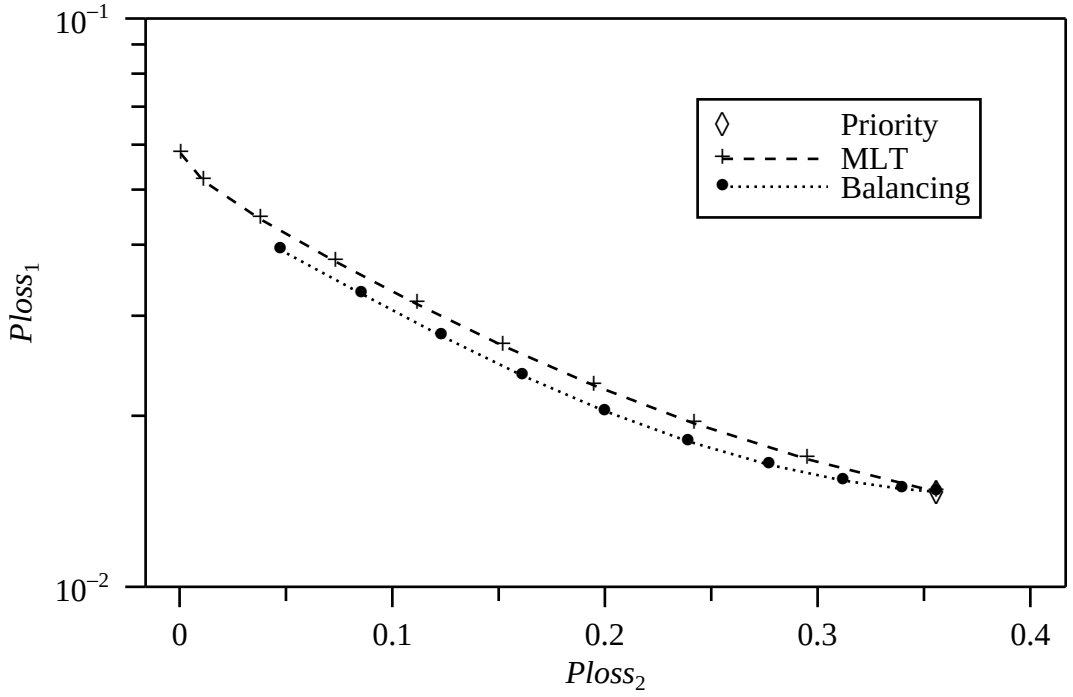

Figure 2.9. $\lambda_1 = 0.75, \lambda_2 = 0.15, r = 10$

laxity thresholds on Class 1 packets, MLT creates a scheduling bias in favor of Class 2 traffic unlike balancing which always maintains a bias in favor of Class 1 traffic. Thus in our scenario, a scheduling bias in favor of Class 2 traffic on top of the high Class 2 arrival rate is sufficient to cause high Class 1 delay loss rates despite the low Class 1 arrival rates. Another interesting observation can be made by considering the values of $Ploss_2$ across the two figures. $Ploss_2$ values are *higher* for $\lambda_2 = 0.15$ than for $\lambda_2 = 0.75$ except for the minimum $Ploss_2$ values under MLT that are lower for lower Class 2 arrival rate. The reason for the higher delay loss rates for lower Class 2 arrival rates lies in the fact that except for low threshold values under MLT, both MLT and balancing have scheduling biases in favor of Class 1 traffic. When the arrival rate of a given class of traffic is low, a scheduling bias in its favor benefits *both* classes of traffic.

2.7 Simulation Study

The results of the previous section were obtained under the assumption of bulk arrivals with geometrically distributed bulk sizes for the two classes of traffic. We also assumed that arrivals were independent from slot to slot. In this section, we relax this independence assumption and study more complex traffic patterns.

The particular traffic model we use follows a traditional approach from the literature [30, 31]. We consider each class of traffic to be a superposition of several voice sources. We model a single voice source as an on-off source with a talkspurt mean of 352 milliseconds and a silence period mean of 650 milliseconds. In our discrete time model, this corresponds to a geometrically distributed number (with mean 22) of voice packets being generated during each talkspurt; each packet is separated by 16 ms. A time slot corresponds to one-third of a millisecond and is normalized to the time taken to transmit a voice packet on a T1 link. Hence, a single source can generate at most one packet every 48 time slots.

We can vary the expected arrival rate of each class of traffic by varying the number of voice sources in each stream. Hence, if we have n superposed sources, the expression for the overall arrival rate λ is given by

$$\lambda = \frac{352 \times n}{48 \times (352 + 650)}.$$

In our study, we set $n = 61$ giving $\lambda_1 = \lambda_2 \approx 0.446$. We obtained delay loss probabilities by averaging over 10 runs with each run going up to 5 million time slots. We discarded the first 9000 slots in each run to account for transient effects. We obtained the 90% confidence intervals for the delay loss values for deadlines of 10 and 30. When the delay loss values are higher (in the order of 10^{-2}), the confidence intervals are tight. For example, for the case of deadline $r = 10$, when $T = 10$ under MLT, the mean value of $Ploss_1$ is 0.0330 and the half-interval length is 0.0005. Similarly, when deadline is 30 and $B = 8$, under balancing the mean value of $Ploss_2$ is 0.0130, the half-interval length is 0.0004. However, when the delay loss probabilities are low (in the order of 10^{-5} or 10^{-6}), the half-interval length is of the same order of magnitude as the delay loss probabilites themselves. For example, when deadline $r = 10$ and $T = 7$, the mean value of $Ploss_1$ under MLT is 0.0000033 and the half-interval length is 0.0000016. Similarly when the deadline is 30 and $B = 5$, the mean value of $Ploss_1$ under balancing is 0.0000056 and the half-interval length is 0.0000035. These confidence interval values suggest to us that we cannot make strong distinctions between MLT and balancing for low loss probabilites– a conclusion that is further supported by the discussion below.

In Figure 2.10 we show the delay loss results for a deadline of 10, and in Figure 2.11 for a deadline of 30. In both figures, for MLT we plot loss probabilites by imposing a threshold on the minimum laxity of Class 1 packets as usual. We then extend the MLT curves by imposing thresholds on the minimum laxity Class 2 packet. This extension is obtained by using a mirror image of the original data points– i.e., by reversing the roles of the two classes of traffic when the two streams are stochastically

identical. We similarly extend the loss probabilities under the balancing scheme as well.

From Figures 2.10 and 2.11, it is evident that for higher deadlines, MLT and balancing become practically indistinguishable. We can also see that the overall loss probabilities become smaller for higher deadlines. The essential conclusion we can draw from this is that for low loss probabilities, complex scheduling disciplines such as MLT or the balancing scheme may not provide much of a tradeoff in terms of performance of both classes of traffic. The easiest solution here would be to assign strict priority to one or the other class of traffic.

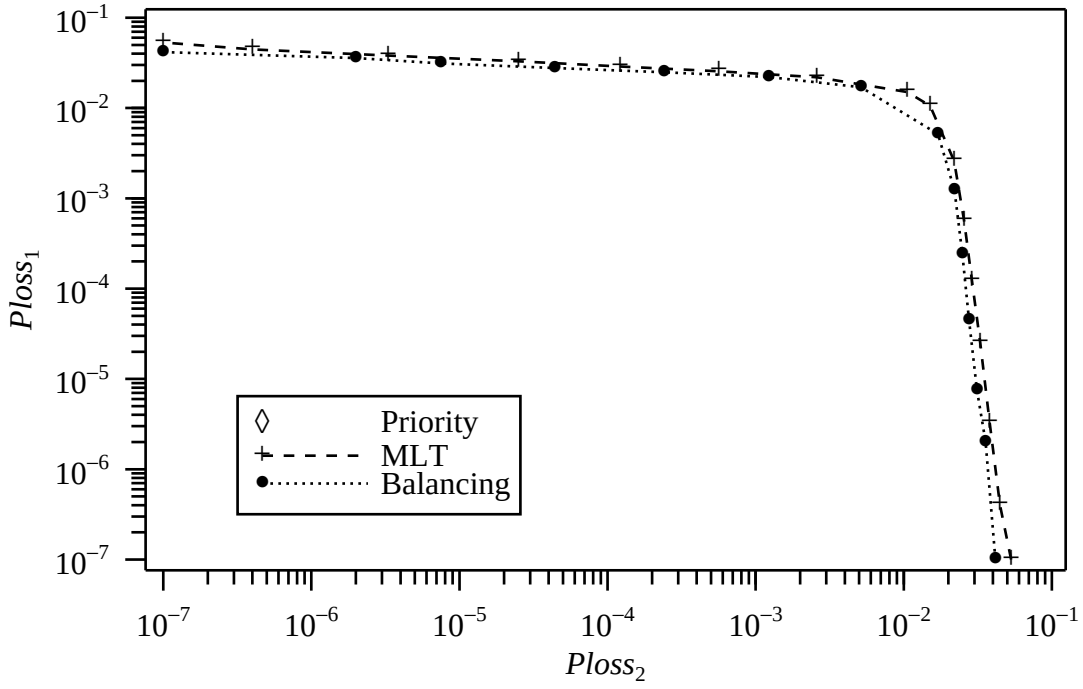

Figure 2.10. Voice sources: $\lambda_1 = 0.446, \lambda_2 = 0.446, r = 10$

2.8 Conclusions

In this chapter, we have considered the problem of scheduling two classes of real-time traffic with correlated time constraints. Three scheduling disciplines were

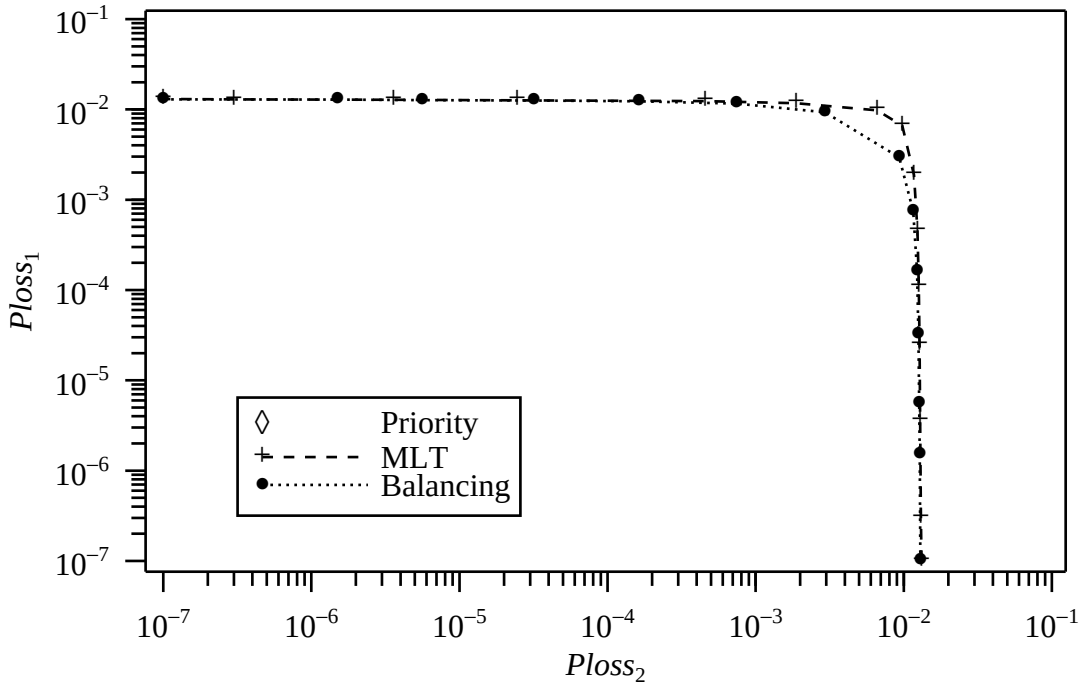

Figure 2.11. Voice sources: $\lambda_1 = 0.446, \lambda_2 = 0.446, r = 30$

studied: a priority discipline which gives strict priority to one class of traffic, a threshold-based scheme in which priority is given to one class of traffic when the minimum laxity of its queued packets falls below some threshold, and a new "balancing" scheme which assigns priority on the basis of the differences in minimum laxities in the two classes of traffic. Our analytic results showed that the balancing discipline, which explicitly considers the difference between minimum laxities of the two classes of traffic, can yield better performance for *both* classes of traffic than MLT in some regions of operation, and can be more effectively used to exploit the tradeoffs that exist between the two classes of traffic. This was found to be particularly the case when time constraints were relatively tight and links were loaded up to an 80 percent nominal load. For loss probabilities that were relatively high (Figure 2.6) or relatively low (Figure 2.7), there was less difference between MLT and balancing.

In carrying out the analysis, we made several assumptions. Since we were interested in the fundamental problem of scheduling two classes of real-time traffic

with specified time constraints, these helped make the analysis easier. For particular applications, however, these assumptions might be hard to justify. There is the assumption of independence between the arrivals of the two classes. With reference to video applications, since both classes are obtained from the same video source, the chances are that there is correlation between them. Further, we assume independence from slot to slot. Again, for video applications, this is difficult to justify. Usually, what is transmitted over the network is the *difference* in the picture from frame to frame with a regular refresh of the entire picture [7]. Hence, there is likely to be correlation from slot to slot. We have relaxed the independence assumptions on traffic arrivals and carried out simulations using models of superposed voice sources as our arrival streams. Our results show that for these types of traffic, there is not much difference between the MLT and balancing disciplines. Further, for low loss probabilities, our best option may be to use the strict priority scheduling policy. This conclusion is similar to the one arrived at when the arrival process was a geomtric bulk arrival process. Hence, another conclusion we may draw is that the particular arrival process does not matter that much in deciding which scheduling discipline to use if steady state delay loss probabilities are low.

CHAPTER 3

COMPARING END-HOST SCHEDULING ALGORITHMS

3.1 Introduction

Many real-time applications can run simultaneously on a multimedia workstation. These applications usually involve the regular reception of real-time data (e.g., voice or video packets) from the network that the workstation is attached to, or the periodic generation of such packets for transmission over the network. Each new packet needs protocol processing and is an arrival to the workstation's CPU. Depending on the scheduling discipline used at the CPU, an arrival may *preempt* the job currently receiving service. These preemptions have associated costs and reduce the amount of effective work that the CPU can do. An interesting problem then is to study how different processor scheduling algorithms perform in terms of the numbers of preemptions that the arrivals cause.

In this chapter, we compare the numbers of preemptions for periodic task-sets under well-known processor scheduling disciplines. We study a dynamic discipline called earliest deadline preemptive scheduling (ED) and static disciplines including the rate monotonic preemptive scheduling algorithm (RM) that have fixed priority assignments, and quantify the superior performance of ED.

Most studies of scheduling algorithms have focused on examining the algorithms' *feasibility* regions. A given task-set, which typically consists of a set of periodic jobs, is said to be feasible under a scheduling discipline if all jobs are able to meet their deadlines (e.g., see [12] and [13, references therein]). Necessary and sufficient conditions for feasibility are now available for testing any periodic task-set with arbitrary deadlines that is scheduled using RM. A feasibility test is not available

for ED when deadlines are arbitrary. However, the fact that any task-set that is schedulable under RM is also schedulable under ED suggests that more work can be extracted out of a processor under ED than under RM. For example, when a task's deadline equals its period, and there is a very large number of classes of jobs, RM can always support processor utilizations up to 69.3%, whereas ED can support utilizations up to 100%. This disparity becomes especially important if we consider the kinds of applications for which we would need to perform workstation scheduling. As an example, consider multimedia conferencing. During a multimedia conference, real-time traffic such as additional voice, video or screen applications, may be randomly added to existing traffic, and thus it may not be possible to limit processor utilization to a bound of less than 70%. Our new results showing that there are always fewer preemptions under ED than under RM, in combination with the fact that ED supports higher utilizations than RM, provide an argument for preferring ED to RM in workstation scheduling.

In the next section of this chapter, we survey some of the work on workstation scheduling performance. We then present a proof of our claim that there are fewer preemptions under ED than under RM. Following this, we discuss simulation results to quantify the difference in the number of preemptions under different scheduling algorithms for well-known task-sets. After this we present simulation results for a more complex scenario in which some operating system costs associated with preemptive algorithms are explicitly modeled.

3.2 Survey of Related Work

The field of scheduling real-time periodic tasks sets has been an active research area for over two decades. Liu and Layland in their classic paper [12] establish the feasibility region for the preemptive rate monotonic algorithm. RM belongs to the class of fixed priority scheduling algorithms. For each policy belonging to this class, each task of a task-set is assigned a static priority level. Jobs of a higher priority level

take precedence over jobs of a lower priority level. In [12] it is demonstrated that the processor utilization bound under RM for scheduling m periodic tasks using RM is $m(2^{1/m} - 1)$ when deadlines equal periods. As $m \rightarrow \infty$, the processor utilization bound goes to 69.3%– i.e., any periodic task-set with processor utilization of less than 69.3% is schedulable using RM. They also demonstrate that if a feasible fixed priority exists for some task-set, the rate monotonic priority assignment is feasible for that task-set, i.e., they establish the optimality of the rate monotonic algorithm in the class of fixed priority scheduling algorithms. In addition to these results, the feasibility bound for the preemptive ED algorithm is shown to be a utilization of 100% in [12].

Following the groundwork that was laid in [12], the field of fixed priority scheduling has seen many advances. An excellent summary of these new results is given in [13]. There now exist results to show the feasibility regions for a periodic task-set with arbitrary deadlines, studies of the average case behavior ([12] gives a worst case analysis), extensions to take care of aperiodic tasks that are mixed in with the periodic tasks and ways to handle preemption overheads and priority inversions. Clearly, our understanding of the behavior of fixed priority scheduling algorithms (including RM) has expanded greatly. However, there has not been any direct comparison of fixed and dynamic scheduling policies from the point of view of the number of preemptions that the policies induce.

An analysis of delay bounds on a single server whose arrival process is a superposition of periodic processes is given by Jean-Marie et. al. [32] for both premptive and nonpreemptive disciplines. A feasibility study based on worst case trajectories for the nonpreemptive case is shown here. The results obtained for a single server queue are also extended to the network case of a slotted unidirectional ring in [32]. Jeffay et. al. [33] also look at nonpreemtive scheduling of a set of periodic or sporadic tasks. They provide a set of necessary and sufficient conditions for such a task-set to

be schedulable and show the optimality of the non-preemptive earliest deadline first algorithm for task-sets that satisfy these conditions.

An argument in favor of a systemic approach to supporting real-time applications in workstations is given in [34]. This involves considering the I/O subsystem and memory management in addition to the issue of processor scheduling. The authors call for creating the necessary hardware such as multiple, concurrent paths to high-bandwidth memory to enable the workstation to run several simultaneous time-critical applications. Bulterman and van Liere [38] show that Unix-like operating systems do not provide much support for real-time applications. The cost of context switching in such environments is considerable. For example, [35] shows that the cost of a *Signal-Wait*, which is the time for a process to signal a waiting process and then to wait on a condition, is about 50 times greater for Ultrix processes than for fast threads.

None of the above work presents a clear and systematic comparison of the number of preemptions seen under various scheduling policies. Our work is interesting in that we provide just such a comparison of the performance of ED and some fixed priority policies for processor scheduling. We *quantify* the relative performance of these disciplines for well-known task sets and provide insight on how the structure of the task-set itself influences the overall number of preemptions. We also explicitly model and simulate some operating system costs involved in preemptive scheduling of task-sets and comment on the time that the CPU spends on these overheads.

3.3 Comparing Numbers of Preemptions under ED and RM: Analysis

We wish to compare the number of preemptions obtained for periodic task-sets scheduled at a single server under the rate monotonic and the earliest deadline first policies. RM assigns higher priority to jobs with lower period, while ED always schedules the job with the closest deadline. We first establish a result for more

general scheduling disciplines and task-sets that need not be periodic, and then apply it to the particular case in which we are interested.

Consider a stream of jobs $u = 1, 2, \ldots$ that arrive at times $a_1 \leq a_2 \leq \ldots$. These jobs belong to one of K classes labelled $k = 1, 2, \ldots, K$. Let κ_u be the class of job u and let σ_u denote its service time. Associated with each class k is a relative deadline r_k. Also associated with each job u is an absolute deadline given by $d_u = a_u + r_{\kappa_u}$. The relative deadline of a job u (r_{κ_u}) is the amount of time from its arrival to when its service must be completed, and the absolute deadline is the time instant by which the job has to complete service. Let π be any scheduling policy that knows all jobs' arrival times, service times and relative deadlines. We introduce the following:

- $N^\pi(t)$ - number of preemptions under policy π by time t, $N^\pi(0) = 0$.
- $V_k^\pi(t)$ - unfinished work of class k jobs under policy π at time t
- c_u^π - the service completion time of job u under policy π.
- π_1 - a non-idling, preemptive policy which gives priority to the job with the smallest relative deadline. By "non-idling" we mean that the server does not remain idle if there is any outstanding work of any class at the server.
- π_2 - a non-idling, preemptive policy which gives priority to the job with the smallest absolute deadline.

Let γ be the non-idling, preemptive policy that always schedules class i jobs when there are no class $j < i$ jobs in the system. Then, Nain and Towsley [36] have established the following lemma:

Lemma 3.1 *For any arbitrary arrival sequence of jobs,*

$$\sum_{k=1}^{i} V_k^\gamma(t) \leq \sum_{k=1}^{i} V_k^\pi(t), \ \forall t: \ 0 \leq t < \infty, \ i = 1, \ldots, K \tag{3.1}$$

We use this result to establish the main theorem of this section.

Theorem 3.1 *If arrival times and service completions never coincide under π_1 and π_2, then for any sequence of service times and arrival times*

$$N^{\pi_2}(t) \leq N^{\pi_1}(t), \ \forall t \geq 0. \tag{3.2}$$

Proof: It suffices to show that for every preemption under π_2 at a time t, there is also a preemption under π_1 at the same time.

Let customer v be preempted under π_2 at time t. Let customer u be under service under π_1 at time t^-. Consider two cases based on the values of the relative deadlines of u and v.

Case 1: $r_{\kappa_u} \geq r_{\kappa_v}$

Since v is preempted at t under π_2, there must be an arrival at time t of some job, say w, such that $d_w < d_v$. This implies that $r_{\kappa_w} < r_{\kappa_v}$ (because $a_w > a_v$). Such an arrival would also preempt u under π_1 at t.

Case 2: $r_{\kappa_u} < r_{\kappa_v}$

Consider two possibilities.

(i) $c_u^{\pi_2} > t$

This condition implies that $d_v < d_u$ and since there is a preemption at t under π_2, we get $d_w < d_v < d_u$. Since $a_u < a_w$, we get that $r_{\kappa_w} < r_{\kappa_u}$ and hence w would preempt u under π_1 at time t.

(ii) $c_u^{\pi_2} < t$

Since job u is still in service at t^- under π_1, we have

$$V_{\kappa_u}^{\pi_2}(t) < V_{\kappa_u}^{\pi_1}(t). \tag{3.3}$$

But, since π_1 is the same policy as γ, we see from Lemma 3.1 that

$$\sum_{i \leq \kappa_u} V_i^{\pi_1}(t) \leq \sum_{i \leq \kappa_u} V_i^{\pi_2}(t). \tag{3.4}$$

It follows from (3.3) and (3.4) that

$$\sum_{i < \kappa_u} V_i^{\pi_1}(t) < \sum_{i < \kappa_u} V_i^{\pi_2}(t). \tag{3.5}$$

Hence there is a non-zero amount of work from some job with a relative deadline (say r_k) smaller than r_{κ_u} that is awaiting service under π_2 at time t. Therefore the arrival

at time t with relative deadline r_{κ_w} must be such that $r_{\kappa_w} < r_k < r_{\kappa_u}$ in order for there to be a preemption at t under π_2. Such an arrival would also preempt job u that is still receiving service under π_1.

Combining Cases **1**, **2(i)** and **2(ii)** yields the inequality shown in (3.2).

Remark: Theorem 3.1 was proven under the assumption that there are no simultaneous events. This assumption is needed to take care of the possibility that there may be a preemption under π_2 at the very instant that there is a service completion under π_1. The arriving job that causes a preemption under π_2 would not cause a preemption under π_1. The way around this is problem is to simply postulate that *all* processing associated with a service completion is completed *before* taking care of an arrival at that instant. Thus, in our scenario, we would also schedule the next job according to the π_1 policy from those jobs that are already in the system before considering the new arrival. The job that has just been scheduled would receive zero service this time around and would be preempted immediately by the new arrival.

We now turn our attention to a periodic task-set with classes $1, 2, \ldots, K$. Jobs of class i have period T_i and the first job of that class arrives at time p_i, $0 \leq p_i < T_i$. We define $P = \{p_1, p_2, \ldots, p_K\}$ as the *initial phasing* of the task-set. The rate monotonic algorithm assigns higher priority to jobs with lower periods. Without loss of generality, we number the tasks so that $T_1 < T_2 < \ldots T_K$ and choose the relative deadlines of jobs to be in the same order relation as their periods. Then, policy π_1 of Theorem 3.1 corresponds to RM. Since policy π_2 assigns priority to the job with the smallest absolute deadline, it corresponds to ED. Hence the following corollary follows directly from Theorem 3.1.

Corollary 3.2 $N^{ED}(t) \leq N^{RM}(t)$, $0 \leq t < \infty$, *for any sequence of arrival times and service times if relative deadlines are monotonic in exact correspondence to the periods in a periodic task-set.*

3.4 Comparing Numbers of Premptions: Simulations

3.4.1 Algorithm Performance Without OS Overheads

Corollary 3.2 establishes a sample path inequality indicating that the number of preemptions under ED is always less than that under RM for any given task-set. In order to determine the actual quantitative difference in the numbers of preemptions for ED and RM, we simulated an avionics task-set [14] shown in Table 3.1 under the two scheduling policies. We simulated the task-set 3000 times with each scheduler for the case of relative deadlines of jobs being equal to their periods, each time with a randomly generated initial phasing, P. The time of the first Class i arrival, p_i, was assumed to be a random variable uniformly distributed in the interval $[0, T_i)$, $i = 1, \ldots, K$. For RM scheduling, all jobs with the same period belong to the same scheduling class; jobs were served in a first-in first-out ($FIFO$) order within each scheduling class. Further, in our simulations we ignored operating system overheads, such as the actual context switching costs, and concentrated on studying the actual numbers of preemptions under the different scheduling algorithms. In the next subsection we present simulation results when some of these overheads are explicitly modeled.

A regeneration cycle for a task set whose jobs have integer periods and an arbitrary initial phasing begins at the greatest idle instant that is less than or equal to the least common multiple of periods of the jobs in the set. The duration of the regeneration cycle will equal the least common multiple of the periods (118000 in the case of the avionics task-set). The notion of a regeneration cycle is important because the sample path behavior of the periodic task-set repeats itself every regeneration cycle. Thus, in our simulations, it was sufffficient to collect statistics during one regeneration cycle for each run.

Let random variables N^{ED} and N^{RM} represent the number of preemptions in a regeneration cycle for ED and RM respectively. Then, random variables (N^{ED}, P)

Table 3.1. Avionics Task Set (in msecs), Util.=83.01%

Execution Time	Period
2.0	25.0
5.0	25.0
1.0	40.0
3.0	50.0
5.0	50.0
8.0	59.0
9.0	80.0
2.0	80.0
5.0	100.0
3.0	200.0
1.0	200.0
1.0	200.0
3.0	200.0
1.0	1000.0
1.0	1000.0

and (N^{RM}, P) denote the numbers of preemptions/cycle for initial phasing P. In our simulations we collected 3000 sample values of (N^{ED}, P) and (N^{RM}, P) and estimated the means of N^{ED} and N^{RM} by the sample means over these 3000 values. We denote these estimated means by $E[N^{ED}]$ and $E[N^{RM}]$. Further, we estimate the mean percentage difference between the number of preemptions under the two disciplines by the mean of the sample differences and denote it as $E[Dif]$. The percentages are measured relative to the number of preemptions under ED.

For the 3000 runs, we obtained $E[N^{ED}] = 9472.21$ and $E[N^{RM}] = 9752.02$. On average, there were 3.04% more preemptions under RM than under ED (relative to ED). More interesting results can been seen by examining the distribution of the percent difference in the numbers of preemptions obtained over one regeneration cycle for RM and ED. Figure 3.1 shows the histogram of the sample values of the percentage differences. As can be seen from the figure, the percentage difference can exceed 21%. Approximately 25% of the runs exhibit a percentage difference that is close to 5% or more. This suggests that for a significant proportion of initial phasings,

there is a non-negligible difference in the numbers of preemptions observed under ED and under RM.

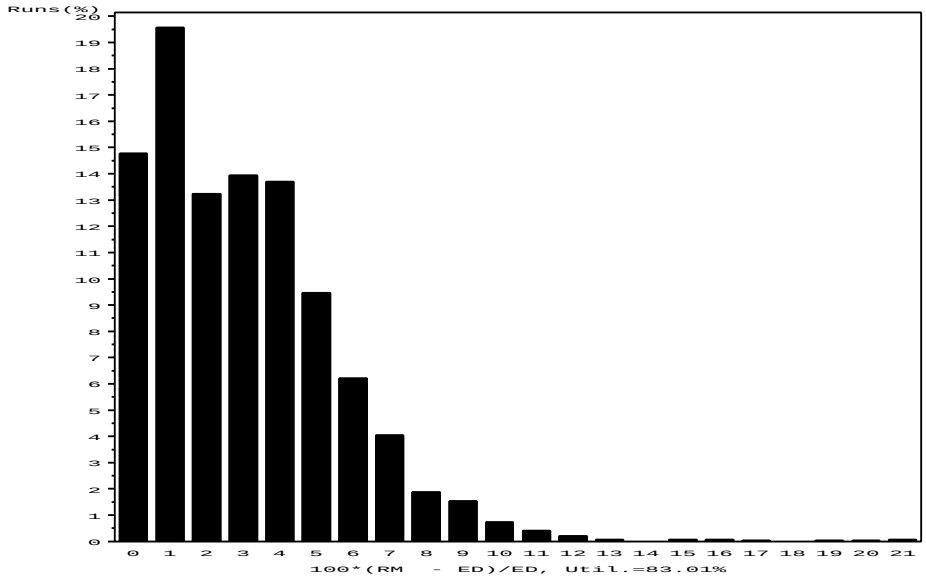

Figure 3.1. Difference in preemptions, RM-ED, (Avionics)

In order to probe how the structure of the task-set affects the number of preemptions, we also simulated a different task-set, INS [14], shown in Table 3.2. This

Table 3.2. INS Task Set (in msecs), Util.=88.44%

Execution Time	Period
11.8	25.0
42.8	400.0
102.8	625.0
202.8	10000.0
1002.8	10000.0
250.0	12500.0

task-set has a regeneration cycle of 50000 (the least common multiple of the periods

of the task-set). In this case, we obtained the average (over 3000 runs) numbers of preemptions to be $E[N^{ED}] = 1614.00$ and $E[N^{RM}] = 1614.02$. This is quite a different result from that obtained for the avionics task-set. A closer look at the behavior of the two task sets on a sample path basis provides us with a plausible hypothesis as to why this is so. The INS task-set is such that the ratio of the highest to lowest periods is 500 (as opposed to 40 for the avionics task-set). Also, the ratio of the maximum to minimum execution times in the INS task-set is over 21 as opposed to only 9 for the avionics task-set. Further, in the INS task-set, jobs belonging to the highest frequency class also have the lowest execution times. This structure of the task-set suggests to us that in the INS task-set, most of the preemptions will be caused by arrivals of the highest frequency jobs. Since both the relative and absolute deadlines of the rapidly arriving high-frequency jobs are likely to be smaller than those of any job in service (and hence the arriving jobs would have higher priority), we would expect preemptions to take place under both ED and RM. An examination of the simulation results does bear this conjecture out, as over 96% of all the preemptions for the INS task-set in both ED and RM are caused by the low-period jobs. By contrast, only around 60% of the preemptions are caused by the highest frequency jobs in avionics task-set under both disciplines.

We obtain further insight into how the structure of the task-set affects the number of preemptions by considering a scheduling algorithm which is only a slight variation of RM. Under $HEHP$ (high execution time, high priority), we assign higher priority to jobs with smaller periods, as in RM scheduling. However, we now make a distinction between jobs that have the same period but different execution times. Among jobs that have the same period, the one with highest execution time receives highest priority. We schedule jobs that have the same period and same execution time in $FIFO$ order. Figure 3.2 shows the histogram for the percent difference in the number of preemptions between $HEHP$ and ED (relative to ED). As can be seen, Figure 3.2 exhibits a very long tail and the percentage difference in the number of preemptions

can be over 188%. About 20% of the runs exhibit a difference in the number of preemptions that is over 8%. We conjecture that the smaller peaks in Figure 3.2 come about as there is a finite probability that jobs of the same period (but of different priorities) arrive close enough to each other that an initial preemption is repeated in a periodic manner. This suggests to us that if we have a task-set with periods that are very close (but not equal), there can be many more preemptions under RM than under ED.

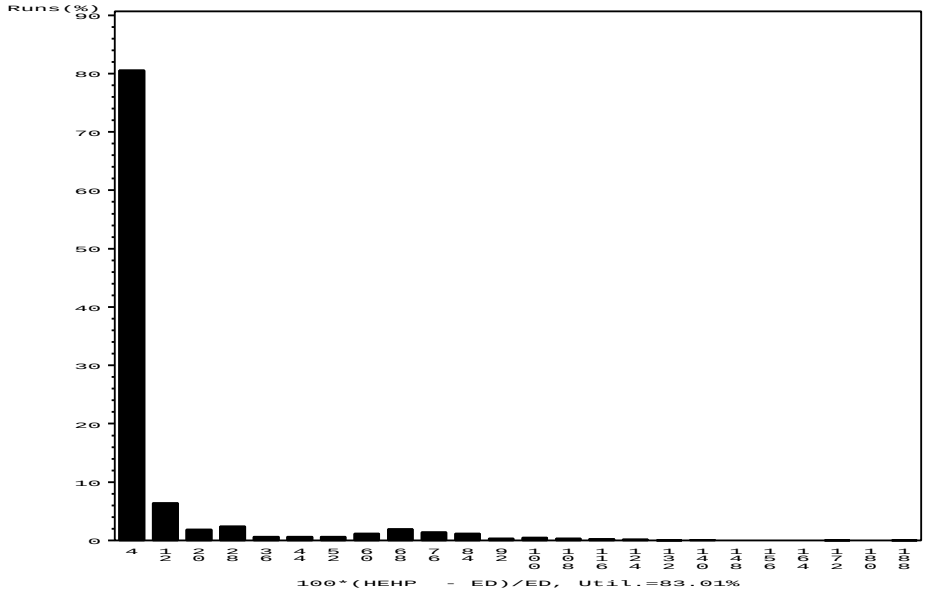

Figure 3.2. Difference in preemptions, HEHP-ED (Avionics)

We also examined how scaling the execution times while keeping the periods the same (and thereby changing the overall utilization) changes the preemption behavior in the case of the avionics and INS task-sets. This corresponds to varying the speed of the processor executing the task-sets. For the INS task-set, reducing all execution times to 90% of their original value reduces the *total* number of preemptions by about 18% for both scheduling disciplines, but has negligible impact on the *relative*

performance of the two schemes. Similarly, increasing the execution times to 110% of their original value increases the total number of preemptions by over 21%, but has negligible impact on the relative performance of the two schemes.

Scaling effects on the avionics task-set are shown in Tables 3.3 and 3.4. Table 3.3 shows the effect of varying the processor utilization on the number of preemptions/run for each of the two scheduling disciplines– RM and ED. Table 3.4 shows the effect of varying processor utilization on the difference between the number of preemptions for the two disciplines for the avionics task set. Both the estimated means and sample maxima (Max$[N^{ED}]$, Max$[N^{RM}]$) of the numbers of preemptions for each of the disciplines (Table 3.3) show a generally increasing trend with the processor utilization. Table 3.4 shows that the average and maximum percentage differences also generally increase as a function of processor utilization (though this increase is not uniform for the maximum percentage differences). The coefficient of variation for any random variable, X is the ratio of its mean to its standard deviation, and the tables also give the respective behaviors of the two disciplines in this regard. The two tables indicate that while there is little change in the coefficient of variation (CV$[N^{ED}]$, CV$[N^{RM}]$) for either discipline with change in processor utilization, the coefficient of variation of the *difference* (CV$[Dif]$) in the numbers of preemptions does decrease with increased utilization. Thus the standard deviation of the difference rises faster than the mean with rising utilization. At higher utilizations, the chance of there being some initial phasing that causes there to be significantly more preemptions under RM than under ED is high- this is seen in the Var$[Dif]$ column in Table 3.4 which shows the variance in the difference in the number of preemptions increases with increasing utilization. We can conclude therefore that at higher utilizations ED is to be even more strongly preferred over RM from the point-of-view of lowering the risk of there being a high number of preemptions. We may further conclude from looking at the difference in the way in which the INS and avionics task-sets respond to scaling that scaling has a more significant effect on more "balanced" task-sets than

on more "skewed" task-sets where the behavior of one class may dominate that of all the other classes.

Table 3.3. Scaling Effects on Algorithm Performance (Avionics)

Util%	Max$[N^{ED}]$	$E[N^{ED}]$	Max$[N^{RM}]$	E$[N^{RM}]$	CV$[N^{ED}]$	CV$[N^{RM}]$
66.41	14015	7441.74	14015	7522.17	0.2345	0.2326
70.56	14320	7934.65	14320	8047.10	0.2299	0.2280
74.71	14648	8674.96	14773	8832.67	0.2196	0.2180
78.86	14900	8940.80	15095	9150.71	0.2224	0.2207
83.01	14182	9472.21	14367	9752.02	0.2201	0.2183
87.16	15054	10164.93	15241	10541.97	0.2216	0.2141
91.31	15649	10847.17	16064	11362.18	0.2152	0.2118
92.14	15752	10959.14	16278	11510.07	0.2152	0.2119

Table 3.4. Scaling Effects on Difference (Avionics)

Util%	$E[Dif]$	Max$[Dif]$	Var$[Dif]$	CV$[Dif]$
66.41	1.14	11.52	2.21	1.3066
70.56	1.48	13.53	3.00	1.1693
74.71	1.88	13.41	3.69	1.0230
78.86	2.42	13.58	4.99	0.9240
83.01	3.04	21.12	6.37	0.8303
87.16	3.82	21.40	7.77	0.7290
91.31	4.90	22.13	9.73	0.6365
92.14	5.18	21.60	10.27	0.6190

We now comment briefly on the results we obtained by studying a multimedia task set described in [37]. We changed some of the periods of the original task-set [37] to obtain a regeneration cycle of manageable duration. This modified task-set shown in Table 3.5 where all the periods are in CPU cycles normalized with respect to the requirements of a 9.6 KHz analog interface controller [37]. The task-set has a nominal utilization of 41.3%. At this utilization, the average difference in the numbers of preemptions between ED and RM was negligible while the maximum difference

across 3000 runs was 3.2%. By scaling the execution times we raised the utilization to 91% and 99% and obtained average differences of 1.5% and 3.3% and maximum differences of 18.9% and 35.7% respectively. Similar observations to those made on the avionics task-set could be made for the multimedia task-set with respect to the coefficient of variation of the difference in the number of preemptions while using ED and RM.

Table 3.5. Multimedia Task Set, Util.=43.1%

Execution Time	Period
0.852	16.0
0.568	16.0
0.284	16.0
0.284	16.0
0.284	16.0
0.047	16.0
0.134	20.0
0.636	24.0
0.562	24.0
0.350	24.0
0.344	24.0
1.136	40.0
4.654	48.0
3.182	128.0
1.625	224.0
0.686	224.0
0.406	224.0
0.261	224.0
0.170	224.0
0.147	224.0
0.144	224.0
0.082	224.0
0.023	224.0
2.165	1120.0

3.4.2 Algorithm Performance With Inclusion of OS Overheads

All the results of Section 3.4.1 were obtained by simulating scheduling disciplines without the inclusion of any system overheads like context switching costs. In this subsection we discuss the numbers of preemptions obtained under different scheduling policies for the avionics task-set shown in Table 3.1 when we account for some of the operating system overheads in our simulations.

In our model, at any instant the server (processor) may be idle, busy or blocking. We further subdivide the blocking state into two as we shall explain shortly.

An arrival to an idle server (i.e., the processor is not doing any work at that time) causes the state to change to busy. On completing service, if there are no other jobs awaiting service, the server idles. If there are waiting jobs, the next one is chosen according to the scheduling discipline at the processor and the server remains in the busy state. There is a cost associated with completion of service for each job called C_{exit} which we model simply as an additional service time requirement for each job.

An arrival to a busy server that has higher priority than the job currently being served causes a context switch with the arriving job incurring a cost of $C_{preempt}$. The server moves into the state "blocked with no higher priority job waiting" for a duration of $C_{preempt}$. If the arrival to a busy server is of lower priority than the job currently receiving service, there is no context switch but there is a cost called $C_{nonpreempt}$ incurred by the job receiving service. The server moves into the state "blocked with no higher priority job waiting" for a duration of $C_{nonpreempt}$. On the completion of the overhead period (whether $C_{preempt}$ or $C_{nonpreempt}$), the job that incurred the cost receives service.

An arrival to a blocked server that has higher priority than the job currently incurring an overhead cost causes the server to move to the state "blocked with a higher priority job waiting". As soon as the current overhead period is finished, there is a context switch to this waiting higher priority job. The server moves back to the state "blocked with no higher priority job waiting" for a duration of $C_{preempt}$.

Arrivals to a blocked server that have lower priority than the job currently incurring an overhead cost are simply enqueued to await service. They do not themselves cause or incur additional overhead.

A diagramatic representation of the above description is given in Figure 3.3. We must point out that this is a simplified model of a processor and that there are other possible overhead costs (such as timer costs) that we do not model.

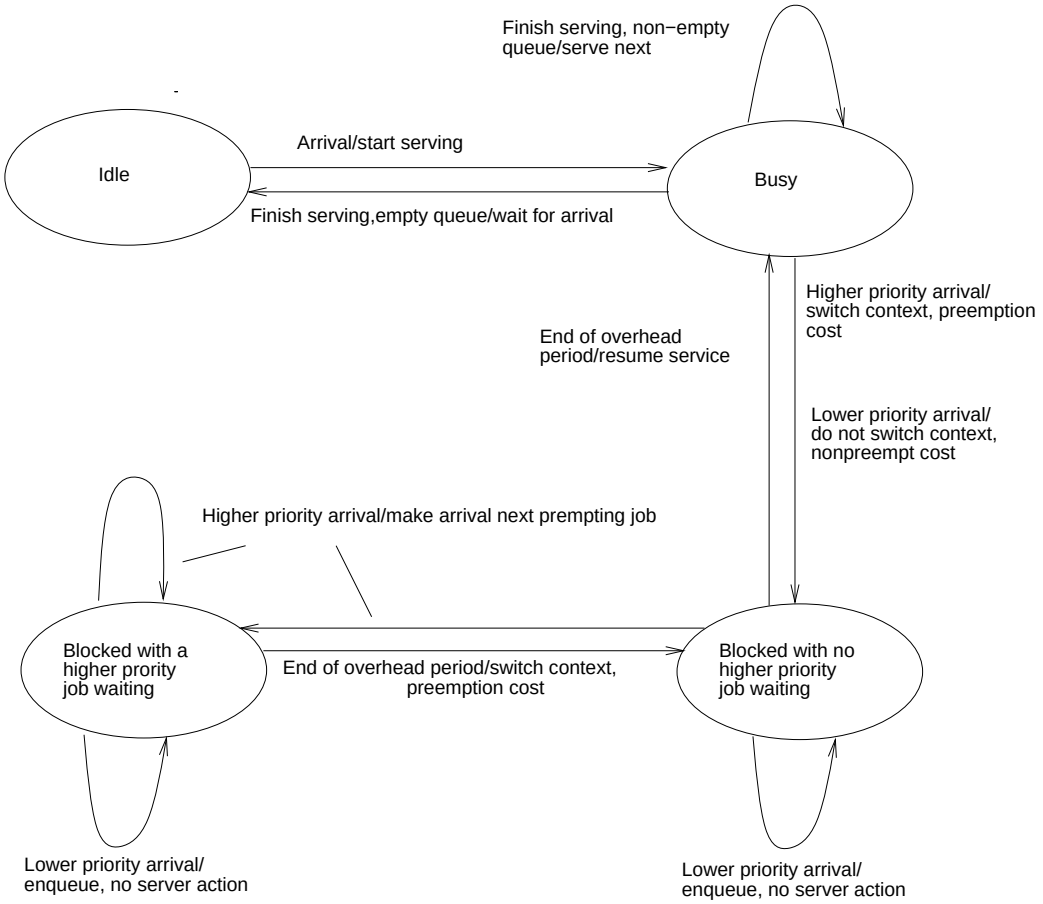

Figure 3.3. State description of processor behavior

The actual values for the preemption and nonpreemption costs are obtained from the work presented in [15] that gives these costs when each job corresponds to an initiation of a real-time thread in the Mach kernel. For the avionics task-set, there

are $n = 15$ threads. The blocking (or nonpreempt) cost, preemption cost and exit cost are given by

$$C_{nonpreempt} = 0.39n + 7.12 = 12.97\mu secs$$

$$C_{preempt} = 0.79n + 30.1 = 41.95\mu secs$$

$$C_{exit} = 0.74n + 28.8 = 39.9\mu secs.$$

We ran 3000 sample runs on the avionics task-set as before and compared the numbers of preemptions under RM and ED when these costs were also included. Tables 3.6 and 3.7 show the results thus obtained when the avionics task-set is used. In Table 3.6 $E[N^{ED}]$ and $E[N^{RM}]$ are the average number of preemptions per cycle, Inc$[N^{ED}]$% and Inc$[N^{RM}]$% are the percentage increases in these averages over the case when there are no explicit costs associated with preemptions (see Table 3.3). Ov.ED% and Ov.RM% are the percents of time the CPU spends processing various overheads for ED and RM respectively. These are obtained by adding all the $C_{preempt}$'s, $C_{nonpreempt}$'s and C_{exit}'s incurred over the 3000 regeneration cycles and determining what percentage of the total simulation time (3000 regeneration cycles) these overheads consitute. All these values are shown for different effective processor utilizations (Util%) which are given in the first column.

We can see immediately that the average numbers of preemptions increase when we take these additional costs into account for both ED and RM. In general, the percentage increase shows an increasing trend with increased processor utilization. Similarly, the percentage time the processor spends on overheads is centered around 1% and shows an increasing trend with increased utilization.

Table 3.7 shows how the difference in the number of preemptions for the two algorithms varies with effective utilization when preemption overheads are included. Here we see that the maximum percent difference over 3000 runs increases as the processor utilizations increase. The conclusions that we may draw from Table 3.7 are very similar to those that may be drawn for Table 3.4 except that the average percent

Table 3.6. Scaling Effects with Overheads (Avionics)

Util%	$E[N^{ED}]$	Inc$[N^{ED}]$%	Ov.ED%	E$[N^{RM}]$	Inc$[N^{RM}]$%	Ov.RM%
66.41	7548.96	1.44	0.89	7635.75	1.51	0.89
70.56	8043.10	1.37	0.92	8167.35	1.49	0.92
74.71	8797.63	1.41	0.95	8968.29	1.54	0.95
78.86	9090.58	1.68	0.97	9317.51	1.82	0.97
83.01	9674.43	2.13	0.99	9981.73	2.36	1.00
87.16	10384.14	2.16	1.02	10796.76	2.42	1.03
91.31	11066.66	2.02	1.06	11644.80	2.49	1.07
92.14	11205.46	2.25	1.06	11827.85	2.76	1.08

Table 3.7. Scaling Effects on Difference with Overheads (Avionics)

Util%	$E[Dif]$	Max$[Dif]$	Var$[Dif]$	CV$[Dif]$
66.41	1.21	11.28	2.37	1.2748
70.56	1.61	13.28	3.28	1.1273
74.71	2.00	13.27	4.00	1.0003
78.86	2.57	17.34	5.36	0.9019
83.01	3.26	19.69	6.69	0.7934
87.16	4.10	23.04	8.30	0.7034
91.31	5.38	22.01	10.77	0.6104
92.14	5.27	27.79	11.72	0.5982

differences are higher when overheads are included. As before, ED is to be preferred to $RM-$ especially when processor utilization are high. Using ED reduces the risk of having large numbers of preemptions.

The difference in the case of including and excluding overheads is not very large primarily because real-time thread management costs are small compared to the execution times of the jobs. Depending on the application, jobs may be run as repeated activations of processes rather than threads. In such situations, we would expect overheads to occupy a much higher proportion of processor resources [35] and consequently, supportable effective CPU utilizations will be lower.

3.5 Conclusions

In this chapter we quantified the difference in the number of preemptions seen when different processor scheduling algorithms are used for periodic task-sets. Our simulation results suggest that in real applications, the preemption overhead induced by RM could be noticeably more than that induced by ED. This could be a serious problem under both real-time [15] and Unix-like [38] operating systems. The actual numbers of preemptions were shown to depend on structure of the task-set used. Because of the periodic nature of the arrivals, there could be a repetitive nature to the preemptions also for certain scheduling algorithms such as $HEHP$.

Preemption performance is one aspect of a scheduling algorithm. Other aspects include feasible regions and ease of implementation. From the points of view of preemption performance and task-set feasibility, ED has been shown to be superior to RM. A common argument in favor of RM has been that static priority policies are easier to implement. A closer examination of the implementation details of ED and RM shows this claim to be questionable. An arrival to an RM scheduler entails interrupt handling and an addition to the tail of a priority queue. An arrival to an ED scheduler involves processing an interrupt and insertion into a deadline-sorted queue. For small numbers of classes of traffic, actual timing differences between these two possibilities are likely to be small. Thus ED is able to meet most criteria for a good workstation scheduling policy.

It must be pointed out that this study limits itself to a particular facet of the workstation performance; processor scheduling algorithm is only one determining factor in many in obtaining good overall real-time performance. The operating system used and the supporting hardware available are also important factors in determining the overall suitability of a given workstation in meeting the requirements of multimedia applications.

CHAPTER 4

SCALABILITY OF RELIABLE MULTICAST PROTOCOLS

4.1 Introduction

Many multimedia applications now involve *multicasting*, the sending of information from one site to a number of other sites simultaneously. Examples of such applications include the so-called shared whiteboard, a distributed and shared screen environment for browsing through or annotating files, and for drawing.

In this chapter we analyse protocols that implement multicasting in a reliable way. Our focus is on studying the scalability of different protocols when host processing power is the bottleneck resource. As indicated in Chapter 1, the prime motivation for studying the scalability properties of different protocols arises from the fact that when there are large numbers of receivers, the use of mechanisms that require sender intervention for reliability could cause the sending host to be overwhelmed by protocol processing overheads.

We begin our discussion by surveying related work. We then describe the three specific reliable multicast protocols and then study and analyze their host throughput performance. We demonstrate how protocols which place the burden for loss-recovery at the receivers (which we call "receiver-initiated" protocols) have better scalability properties than protocols that place the burden of processing at the sender, both through the derivation of complexity expressions and with numerical results on particular workloads.

4.2 Survey of Related Work

In this survey we examine previous work in area of reliable multicast protocols. Some of this work has to do with multicast routing and some with the analysis

of various sender-initiated and receiver-initiated protocols for point-to-multipoint communication.

Multicasting in a local area network (LAN) environment entails taking advantage of the broadcast routing properties of a LAN. Deering and Cheriton [23] provide extensions to the distance-vector and link-state routing algorithms to support multicasting beyond a single LAN. They describe how hierarchical multicast routing can be used for multicasting in very large internetworks. They also describe how new users can join and leave a multicast session and the functionality needed at bridges and routers to support their algorithms. An actual experiment using IP multicast routing is described in [19]. Since the necessary host extensions to support multicasting are not yet available on most Internet routers, this experiment was carried out using the experimental DARTnet as a backbone. The work described in [23] is essentially centered on network-related issues for multicasting and does not consider host-related issues such as processing requirements.

Many papers specify different kinds of protocols for reliable multicast. For example, Chang and Maxemchuck [39] describe a reliable broadcast protocol. Their protocol designates a single primary receiver called the token site. This site has the responsibility of acknowledging all broadcast messages and responding to any retransmission requests from other receivers. This responsibility is rotated among the various receivers. Since only one receiver responds to broadcast messages, only one acknowledgment is received per message.

Several papers analyze multicasting variations of well-known ARQ error recovery schemes such as stop-and-wait, go-back-N and selective repeat. Towsley [40] looks at the go-back-N error control protocol for the point-to-multipoint case. Each message is either ACKed or NAKed. Expressions are derived for message delays. Other work on the analysis of go-back-N multicast protocols includes that given in [41, 42, 43]. Wang and Silvester [41] provide throughput analyses for stop-and-wait and go-back-N schemes at the data link layer. The protocols discussed in [41] are generalizations of

the go-back-N protocols given in [42]. It is proposed in [41] that multiple copies of a data frame be transmitted to a broadcast channel instead of a single copy in order to increase throughput. A dynamic programming optimization technique is used to determine the optimal number of copies of the data frame to be sent, based on round-trip delays and error probabilities. Throughput expressions are derived in [41] for this scenario and expressions for the average delay are given by the same authors in [43]. While the variation of throughput with number of recievers is given in [41], no attempt is made to characterize host processing costs.

Towsley and Mithal [44] give an analysis for the selective repeat ARQ where the receivers have finite buffers. This work builds on an earlier analysis of the point-to-point selective repeat ARQ by Weldon [45] and improves on the earlier work by obtaining a tighter lower bound on the maximum throughput. It is demonstrated here that for low bit error rates (10^{-5} or less) the performance of this protocol approaches that of the idealized selective repeat ARQ. While results showing the variation of throughput with the number of receivers for different protocols are given in [44], the maximum throughput is obtained simply as the inverse of the expected number of transmissions for successful receipt of a packet at all receivers. Ram Chandran and Lin [46] apply the dynamic programming optimization technique given in [41] to selective repeat ARQ and carry out a throughput analysis. In both [44] and [46] there is no attempt to explicitly model host processing requirements. Shacham and Towsley also consider the selective repeat ARQ in [47]. This work is primarily concerned with obtaining the distributions of packet resequencing delay and buffer occupancy at the receiver.

The throughput performances of go-back-N, selective repeat with infinite buffers and selective repeat with finite buffers are compared for two different retransmission strategies by Sabnani and Schwartz [48]. The target environment is a satellite broadcast channel and the transmitter is modeled as a queue. It is shown in [48] that only

selective repeat with infinite buffers provides acceptable performance when positive acknowledgements are used for error control.

All the work cited above on ARQ protocols is based on *sender-initiated* methods, i.e., if an ACK or a NAK is not received within a timeout interval, the sender initiates retransmissions. Ramakrishnan and Jain [49] designed and evaluated, through analysis and simulation, a receiver-initiated window based reliable multicast protocol suitable for a local area network. Jacobson has used similar ideas to implement a reliable multicast protocol suitable for the Internet [50].

For wide area networks (WANs), Yavatkar and Manoj [21] describe an end-to-end transport protocol for multimedia multicast. The authors evaluate approaches based on combinations of redundant transmissions, rate-based flow control and negative acknowledgments. They provide a simulation study to show that simple optimistic strategies work well in scaling to hundreds of receivers. Their main interest is in meeting a prespecified QOS requirement (loss over a time epoch) and there is not much discussion of how various policy decisions impact host processing costs.

Our work differs from all of the above in that our primary focus is on the host processing requirements of reliable multicast protocols and that we are interested in the scalability of these protocols. This is an important consideration as it is possible that the protocol processing overhead associated with error-recovery may overwhelm the sender in the one-to-many scenario. This is the *implosion* problem at the sender that we discussed in Chapter 1. It is necessary, therefore, to study how different protocols behave as the number of participants in a multicast session increases. In our work we explicitly model the various components of host processing necessary for reliable end-to-end multicast communication and compare the scalability of different protocols. To this end, we consider the fundamental features of sender-initiated and receiver-initiated protocols rather than attempt an elaborate characterization of all the possible nuances of these complex error-recovery mechanisms.

4.3 Protocol Description

We now describe the two approaches for reliable transmission of data from a sender to multiple receivers. As noted above, the sender-initiated approach, based on the use of ACKs, places much of the burden for ensuring reliable packet transmission on the sender whereas the receiver-initiated approach, based on NAKs, shifts this burden to the receivers. The section ends with a description of our network model. In the remainder of this chapter, the term "broadcast," when used, will refer to the transmission of a packet to all stations involved in the multicast communication.

4.3.1 Sender-Initiated Protocols

A sender-initiated protocol requires the sender to maintain a list (called the ACK list), for each packet, of the receivers from which it has received a positive acknowledgment (ACK). Each time a receiver correctly receives a packet, it returns an ACK. Upon receipt of the ACK, the sender updates the ACK list for the corresponding packet. Lost packets are dealt with through the use of timers. Specifically, the sender starts a timer at the time of a packet transmission and, if it expires prior to the sender having received ACKs for this packet from every receiver, the sender retransmits the packet and restarts the timer.

Most traditional error-recovery protocols (e.g., TCP, HDLC, TP4) and many early multicast/broadcast protocols [40, 48, 42, 49, 44] are based on this approach. Rather than focus on a specific protocol, we will instead focus on a *generic* protocol which exhibits the following behavior.

- error recovery is selective repeat, i.e., only packets that are suspected to be lost or corrupted are retransmitted,
- anytime that a sender transmits or retransmits a packet, it broadcasts to all receivers and starts a timer,

- each time that a receiver receives a packet correctly, it transmits an ACK to the sender over a point-to-point connection,
- whenever a timer expires, the sender rebroadcasts the corresponding packet, giving it priority over new packets.

This will be referred to as protocol (A).

This protocol can be optimized in numerous ways, such as grouping ACKs for different packets into a single control packet (see [40, 48] for examples of such optimizations). We will not pursue them in this dissertation.

4.3.2 Receiver-Initiated Protocols

A receiver-initiated protocol places the responsibility for ensuring reliable packet delivery on the receivers. The sender continues to transmits new data packets until it receives a negative acknowledgment (NAK) from a receiver. When this occurs, the sender then retransmits (i.e., again multicasts) the packet required by that receiver. The role of the receiver is to check for lost packets. If it decides that it has not received a particular packet, it transmits a NAK to the sender. In order to guard against either the loss of the NAK or the subsequent packet retransmission, the receiver uses timers in a manner similar to the way the sender uses them in sender-initiated protocols. Typically a receiver will detect a lost packet when it receives packets with larger sequence numbers (or after a timeout if it is expecting a retransmission). In the case that the sender does not always have packets to send (i.e., must occasionally wait for data to be produced at a higher level), it may be necessary for the sender to multicast periodic state information (e.g., giving the sequence number of the last transmitted packet) while it is idle; we ignore this potential overhead in our ensuing analysis, as we are primarily interested in determining the maximum throughput of the protocols.

We will focus on a generic receiver-initiated protocol (N1) that exhibits the following behavior:

- the sender broadcasts all packets,
- the sender gives priority to retransmissions over transmissions of new packets,
- whenever a receiver detects a lost packet, it transmits a NAK to the sender over a point-to-point channel and starts a timer,
- the expiration of a timer without prior reception of the corresponding packet serves as the detection of a lost packet.

We also study the following variation (N2) suggested by Ramakrishnan and Jain [49] for a LAN and Jacobson [50] for a WAN

- the sender broadcasts all packets and state information,
- the sender gives priority to retransmissions over transmissions of new packets,
- whenever a receiver detects a lost packet, it waits for a random period of time and then broadcasts a NAK to the sender and all other receivers, and starts a timer,
- upon receipt of a NAK for a packet which a receiver has not received, but for which it initiated the random delay prior to sending a NAK, the receiver sets a timer and behaves as if it had sent that NAK,
- the expiration of a timer without prior reception of the corresponding packet serves as the detection of a lost packet.

This second protocol attempts to ensure that at most one NAK is returned to the sender per packet transmission by staggering the generation of NAKs and broadcasting them to all other receivers. Ideally, the first NAK generated for a given packet should arrive at all other receivers prior to their generating additional NAKs. As with the sender-initiated protocols, there are numerous optimizations possible which we will not explore in this dissertation (see [49] for an example).

4.3.3 Network and Error Model

We assume that one sender transmits a continuous stream of packets to R receivers. We further assume that all loss events at all receivers for all transmissions are mutually independent and that the probability of packet loss, p, is independent of receiver. We further assume that ACKs/NAKs are never lost.

The assumption that losses occur as independent events does not typically hold as different receivers will share portions of the multicast tree connecting all of the receivers and sender. Consequently the loss or corruption of a packet on a particular link will affect all receivers sharing that link. However, we expect our analyses to produce pessimistic bounds on throughput as a consequence of this independence assumption. The assumption that ACKs/NAKs are never lost is reasonable as control packets are small and are typically given better treatment in a network. If desired, this assumption can be relaxed by following the analysis given in [40].

4.4 Throughput Analysis of Sender-Initiated Protocols

We consider a single sender broadcasting to R identical receivers. As the behavior of the sender differs from that of a receiver, we consider their behaviors separately. Throughout our analysis we will introduce a number of different parameters as we need them. These are collected in Table 4.1 for ease of lookup. We consider the sender first.

In order to compute the maximum supportable throughput, we will determine the processing time required by the sender to *successfully* multicast a randomly chosen packet to all receivers. Let us denote this processing requirement by X^A. The sender must first obtain the data from a higher level protocol/application. A packet containing this data is constructed and transmitted and a timer is set. Following this, the sender must process every ACK received for the packet. In addition, each time that the timer expires and the sender has not received ACKs from all receivers for

Table 4.1. Notation

X_f	-	the time to feed in a new packet from the higher protocol layer.
X_p	-	the time to process the transmission of a packet.
X_a, X_n, X_t	-	the times to process an ACK, a NAK and a timeout at the sender respectively.
Y_p, Y_t, Y_f	-	the times to process a newly received packet, a timeout, and to playout a packet at a receiver.
Y_a, Y_n	-	the times to process and transmit an ACK and a NAK from the receiver respectively.
Y_n'	-	the time required to receive and process a NAK at a receiver.
p	-	the probability of loss at a receiver; losses at different receivers are assumed to be independent events.
L_r^w	-	the number of NAKs from receiver r, $w \in \{N1, N2\}$.
L^w	-	the total number of NAKs from all receivers per packet, $w \in \{N1, N2\}$.
L_r^A	-	the number of ACKs from receiver r.
L^A	-	the total number of ACKs from all receivers per packet.
M_r	-	the number of transmissions necessary for receiver r to successfully receive a packet.
M	-	the number of transmissions for all receivers to receive the packet correctly; $M = \max_r \{M_r\}$.
X^w, Y^w	-	the processing time per packet at the sender and receiver respectively in protocol $w \in \{A, N1, N2\}$.
Λ_s^w, Λ_r^w, Λ^w,	-	the throughput for protocol $w \in \{A, N1, N2\}$ at the sender, receiver and system.

this packet, the packet must be rebroadcasted and the timer restarted. Given these considerations, we have

$$X^A = X_f + X_p(1) + \sum_{m=2}^{M} (X_t(m) + X_p(m)) + \sum_{i=1}^{L^A} X_a(i) \tag{4.1}$$

where X_f is the processing requirement for receiving the data from the higher layers, $X_p(m)$ is the packet processing requirement associated with the m-th transmission of a given packet, $X_t(m)$ is the requirement for processing the timer interrupt that will result in the mth transmission of the packet, $X_a(i)$ is the processing requirement for the i-th ACK received for this packet, L^A is the total number of ACKs received for the packet, and M is the total number of transmissions required to transmit the packet correctly to all R receivers. These processing times include the times required

to perform a context switch along with processing costs specific to the operation. For example, $X_a(i)$ includes the processing required to take the packet up the protocol stack to the transport layer (or higher). It also includes the time required to update the ACK list and, possibly, to turn off the timer.

We assume that the processing requirements have general distributions, that they are independent of each other, and that $\{X_t(m)\}$, $\{X_p(m)\}$ and $\{X_a(i)\}$ are each identically distributed sequences of r.v.'s. Henceforth, we omit the arguments m and i. The statistics for the r.v.'s L^A and M will be obtained shortly.

We are interested in the mean processing requirements per packet in order for the sender to broadcast packets reliably to all of the receivers. It is given by

$$E[X^A] = E[M]E[X_p] + (E[M]-1)E[X_t] + E[L^A]E[X_a] + E[X_f]. \quad (4.2)$$

Consider the r.v. L^A. Define L_r^A to be the number of ACKs generated by receiver r. Now, $E[L_r^A] = E[M](1-p)$ is the expected number of ACKs generated by r. As the receivers are statistically identical, we have

$$E[L^A] = RE[M](1-p). \quad (4.3)$$

Note that we have assumed here that a receiver generates an ACK for each successfully received packet, regardless of whether it has already previously acknowledged the packet. Substituting this last expression into (4.2) yields the following expression for $E[X^A]$,

$$\begin{aligned} E[X^A] &= E[M]E[X_p] + (E[M]-1)E[X_t] + RE[M](1-p)E[X_a] \\ &\quad + E[X_f]. \end{aligned} \quad (4.4)$$

The only unknown in the right hand side of equations (4.3) and (4.4) is $E[M]$. Define M_r to be the number of transmissions required for receiver r to receive the packet correctly. Now, $P[M_r \leq m] = 1 - p^m$, $m = 1, \ldots$ and $E[M_r] = 1/(1-p)$. As the events of lost packets at different receivers are independent, we have

$$P[M \leq m] = \prod_{r=1}^{R} P[M_r \leq m]$$

$$
\begin{aligned}
&= (1-p^m)^R \\
&= \sum_{i=0}^{R} \binom{R}{i} (-1)^i p^{im}. \qquad (4.5)
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
P[M=m] &= P[M \le m] - P[M \le m-1], \quad m = 1, 2, \ldots \\
&= \sum_{i=0}^{R} \binom{R}{i} (-1)^i p^{i(m-1)} (p^i - 1) \qquad (4.6)
\end{aligned}
$$

and we can write the mean of the above distribution as

$$
\begin{aligned}
E[M] &= \sum_{m=1}^{\infty} m P[M=m] \\
&= \sum_{m=1}^{\infty} m \sum_{i=0}^{R} \binom{R}{i} (-1)^i p^{im} (1 - p^{-i}) \\
&= \sum_{i=0}^{R} \binom{R}{i} (-1)^i (1 - p^{-i}) \sum_{m=1}^{\infty} m p^{im} \\
&= \sum_{i=1}^{R} \binom{R}{i} (-1)^{i+1} \frac{1}{(1-p^i)} \qquad (4.7)
\end{aligned}
$$

$E[M]$ can be obtained numerically from (4.7). The above expression becomes difficult to evaluate for larger values of R because of the alternating signs. The following is an approximation for $E[M]$ for large R (> 35):

$$
E[M] \approx a + \frac{(H_{35} - H_R)}{\ln(p)} \qquad (4.8)
$$

where a is the value of $E[M]$ for $R = 35$ and $H_k = \sum_{i=1}^{k} 1/i,\ k \ge 1$, is the harmonic series.

If we let Λ_s^A denote the rate at which the sender can successfully transmit packets to all receivers then $\Lambda_s^A = 1/E[X^A]$.

In a similar fashion, the mean processing requirement at the receiver for a randomly chosen packet is

$$
E[Y^A] = E[M](1-p)E[Y_p] + E[M](1-p)E[Y_a] + E[Y_f].
$$

Here the first term corresponds to the mean time spent processing packets received, the second term corresponds to the mean time spent producing and transmitting

acknowledgments, and the third term corresponds to the mean time required to pass the data up to the next level. In the above equation, $E[M]$ is as given in (4.7). The maximum packet processing rate at the receiver is $\Lambda_r^A = 1/E[Y^A]$.

The overall system throughput, Λ^A, for this scheme is given by the minimum of the per-packet processing rates at the sender and at the receiver, i.e.,

$$\Lambda^A = \min\{\Lambda_s^A, \Lambda_r^A\}. \tag{4.9}$$

It can be shown that E[M] is $O(1+\ln R/(-\ln p))$ (see Appendix B). Hence, $E[X^A]$ is $O(R(1+\ln R/(-\ln p)))$ and $E[Y^A] = O(1-p+(1-p)\ln R/(-\ln p))$. Observe that, as $p \to 0$, $E[X^A] \to O(R)$. Consequently, this protocol places a significant processing burden on the sender for large values of R even in the case of a highly reliable system. For constant p, $E[X^A]$ is $O(R \ln R)$.

4.5 Throughput Analysis of Receiver-Initiated Protocols

We analyze (N1) first by considering the sender. Let X^{N1} denote the mean packet processing requirement at the sender under the receiver-initiated protocol (N1). This processing time can be expressed as

$$X^{N1} = X_f + \sum_{m=1}^{M} X_p(m) + \sum_{i=1}^{L^{N1}} X_n(i) \tag{4.10}$$

where the first term corresponds to the processing time required to obtain the data from the higher layers, the second term corresponds to the processing time associated with the M different transmissions of the packet and the third term corresponds to the processing time for the NAKs that are transmitted from the receivers to the sender. As before, we make no assumptions regarding the distributions of these r.v.'s except that they are independent of each other and $\{X_p(m)\}$ and $\{X_n(i)\}$ are sequences of identically distributed r.v.'s. As before, M is the number of transmissions required and L^{N1} is the number of NAKs received.

The mean processing time is given as

$$E[X^{N1}] = E[X_f] + E[M]E[X_p] + E[L^{N1}]E[X_n].$$

Now, the number of NAKs returned to the sender by receiver r is $M_r - 1$ with mean $p/(1-p)$. Hence the mean number of NAKs returned by all receivers is $E[L^{N1}] = Rp/(1-p)$ and the mean per packet processing time at the sender is

$$E[X^{N1}] = E[X_f] + E[M]E[X_p] + RpE[X_n]/(1-p) \tag{4.11}$$

and the sender packet processing rate is $\Lambda_s^{N1} = 1/E[X^{N1}]$.

We focus next on the mean per packet processing time at a receiver. Similar to the analysis leading to (4.2), we have

$$\begin{aligned} E[Y^{N1}] &= E[Y_f] + E[M](1-p)E[Y_p] \\ &\quad + P[M_r > 1](E[M_r|M_r > 1] - 1)E[Y_n] \\ &\quad + P[M_r > 2](E[M_r|M_r > 2] - 2)E[Y_t]. \end{aligned}$$

Here the first two terms correspond to the processing required to correctly receive packets and send them to the next layer. The third term corresponds to the processing required to prepare and return NAKs. Note that this only occurs each time the receiver determines the packet to be lost *prior to the first correct receipt of this packet.* The last term corresponds to processing of the timer when it expires. Again, this is only required after the first transmission (if lost) up to, but not including, the first correct reception of a given packet.

From the distribution of M_r, it follows that

$$\begin{aligned} E[M_r|M_r > 1] &= (2-p)/(1-p), \\ E[M_r|M_r > 2] &= (3-2p)/(1-p). \end{aligned}$$

Substitution into (4.12) yields

$$E[Y^{N1}] = E[Y_f] + E[M](1-p)E[Y_p] + pE[Y_n]/(1-p) + p^2E[Y_t]/(1-p)$$

and the receiver packet processing rate is $\Lambda_r^{N1} = 1/E[Y^{N1}]$.

Finally, the throughput Λ^{N1} is given by

$$\Lambda^{N1} = \min\{\Lambda_s^{N1}, \Lambda_r^{N1}\}.$$

For this protocol we have $E[X^{N1}] = O(1+pR/(1-p))$ and $E[Y^{N1}] = O(1-p+(1-p)\ln R/(-\ln p))$. Consequently, it is better suited to large scale multicasts than the generic ACK-based protocol (A). Observe that $E[X^{N1}] = O(1)$ and $E[Y^{N1}] = O(1)$ when $p \to 0$. For constant p, $E[X^{N1}]$ is $O(R)$.

We end this section with the analysis of the receiver-initiated protocol (N2). Protocol (N2) differs from (N1) in two ways. First, NAKs are broadcast from a receiver to all other receivers as well as to the sender. If a receiver receives a NAK for a packet that it has not received correctly prior to sending its own NAK, then it need not return a NAK. The addition of a random delay at a receiver prior to returning a NAK ensures that *most of the time* only one NAK will be generated among all of the receivers and that it will act as a signal to the remaining receivers to not return a NAK. How well this mechanism works to limit the number of NAKs per transmission depends on the network topology and the choice of the random delay distribution. As we are solely concerned with throughput, we will assume that the delays are sufficiently long that the event of two or more NAKs being generated by a transmission is sufficiently small to be ignored.

With these modifications, it is easy to see that

$$1/\Lambda_s^{N2} = E[X^{N2}] = E[X_f] + E[M]E[X_p] + (E[M]-1)E[X_n] \qquad (4.12)$$

and

$$\begin{aligned} 1/\Lambda_r^{N2} &= E[Y^{N2}] = E[Y_f] + E[M](1-p)E[Y_p] \\ &\quad + (E[M]-1)(E[Y_n]/R + (R-1)E[Y_n']/R) \end{aligned}$$

$$+ P[M_r > 2](E[M_r|M_r > 2] - 2)E[Y_t].$$

Last,

$$\Lambda^{N2} = \min\{\Lambda_s^{N2}, \Lambda_r^{N2}\}.$$

Note that for (N2) we have at the sender $E[X^{N2}] = O(1 + \ln R/(-\ln p))$ and at the receiver $E[Y^{N2}] = O(1 - p + (1 - p)\ln R/(-\ln p))$. This protocol shows the best potential for handling multicasts to large receiver groups as for constant p, $E[X^{N2}]$ is $O(\ln R)$. Furthermore, $E[X^{N2}] = E[Y^{N2}] = O(1)$ when $p \to 0$.

4.6 Numerical Results

4.6.1 Equal Processing Costs for Components of Host Protocol

We now examine the relative performances of protocols (N1), (N2) and (A). For ease of comparison, we set $E[X_p] = E[X_t] = E[X_a] = E[X_n] = E[X_f] = E[Y_p] = E[Y_t] = E[Y_a] = E[Y_n] = E[Y_f] = 1$. The processing associated with each of these quantities each involves interrupt costs (interrupt overhead and context switching costs), possibly some data copying, and a small amount of control action. Hence, the above simplification is reasonable when packet sizes and copying costs are small. For all the figures, $E[M]$ is obtained directly from the expression in (4.7) up to the value of $R = 35$ and from the approximation given in (4.8) for $R > 35$.

Figure 4.1 shows how the ratio of sender throughputs obtained under (N1) and (A) varies with the number of receivers for different loss probabilities. It will become clear later that the sender throughputs are the same as the system throughputs as the sender is the bottleneck under all three protocols (A), (N1) and (N2). As can be seen from the figure, there is no situation in which the throughput ratio falls below 1.0, i.e., (N1) always outperforms (A) in this regard. As may be expected, when the loss probability is low (say 0.01), the relative performance of (N1) is better than when the loss probability is high (say 0.50). This is because a receiver-initiated scheme places a very light burden on the sender for low loss probabilities while a sender-initiated

scheme generates many acknowledgments that need to be processed at the sender. For a given loss probability, the relative performance of (N1) improves over that of (A) with increasing R. This improvement is logarithmic as indicated in the complexity results presented in the previous section.

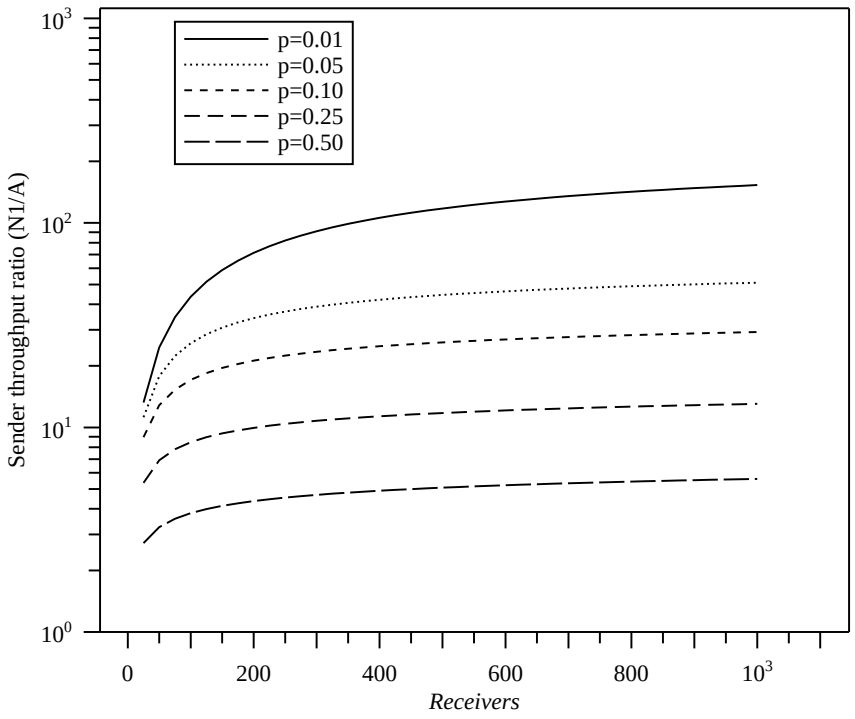

Figure 4.1. Ratio of sender throughputs for (N1) and (A): $\Lambda_s^{N1}/\Lambda_s^A$ vs R

Figure 4.2 shows the relative performance of (N1) and (A) at the receiver end. In this case too, for loss rates of up to 50%, (N1) provides better performance than (A). Though not shown in the figure, it is true that for loss rates higher than this, (A) provides better performance at the receiver when the number of receivers is low. As can be seen from the figure, for the same loss probability, the throughput ratio begins to reach an asymptotically constant value. This is as expected as both $1/\Lambda_r^{N1}$ and $1/\Lambda_r^A$ are $O(\ln R)$ for constant p.

Figure 4.3 shows the relative throughput performance at the sender for (N2) and (A). The rise in performance of (N2) as a function of R relative to (A) is much steeper

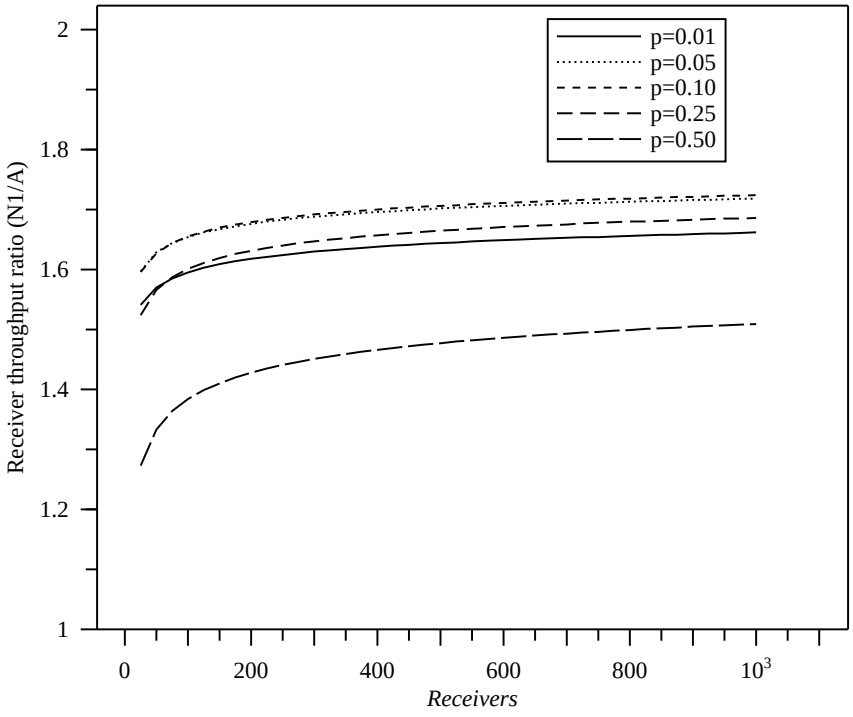

Figure 4.2. Ratio of receiver throughputs for (N1) and (A): $\Lambda_r^{N1}/\Lambda_r^A$ vs R

than that of (N1) as can be seen by comparing curves across Figures 4.1 and 4.3. This is so because under (N2) there is only at most one NAK per transmission that has to be processed at the sender. Protocol (N2) is also relatively less sensitive to the loss probability p than (N1). Both schemes, however, clearly outperform (A). A direct comparison between the performances at the sender of (N2) and (N1) is shown in Figure 4.4. As can be seen from this figure, at low loss probabilities and for low numbers of receivers, the performance of (N2) and (N1) is close. As $p \to 0$, the two schemes become identical. However, for higher p and R, (N2) does far better than (N1). Since the main aim in a multicast environment would be to reduce the load at the sender, we can conclude from Figures 4.1, 4.3 and 4.4 that (N2) provides us with the best relative performance and is to be preferred over (A) or (N1).

Figure 4.5 shows the relative receiver performance of (N2) and (A) for different values of p. As can be seen from the graphs, for high values of loss probability, (A) performs better than (N2). This is so because a greater part of the processing

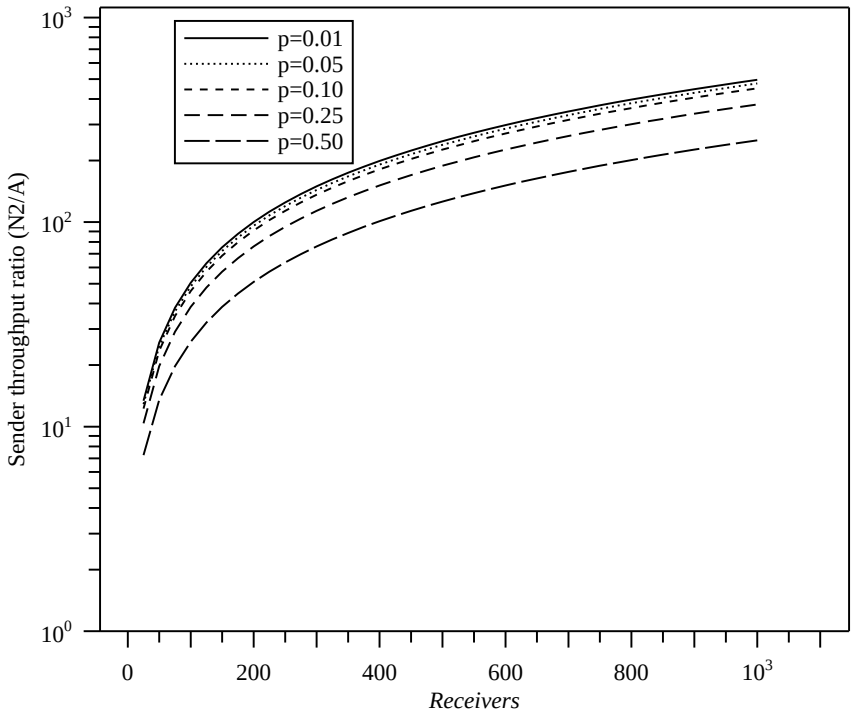

Figure 4.3. Ratio of sender throughputs for (N2) and (A): $\Lambda_s^{N2}/\Lambda_s^A$ vs R

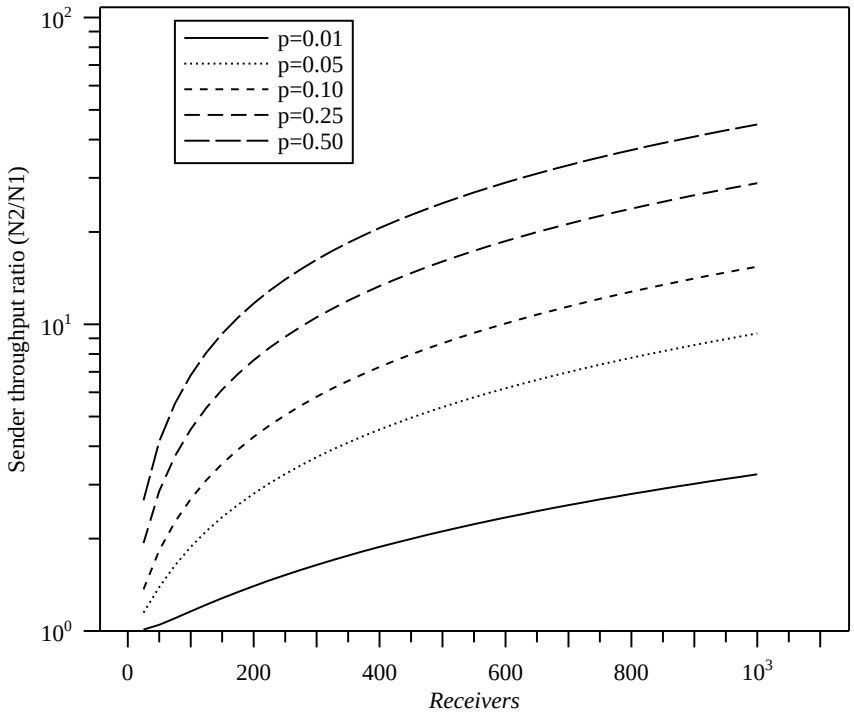

Figure 4.4. Ratio of sender throughputs for (N2) and (N1): $\Lambda_s^{N2}/\Lambda_s^{N1}$ vs R

complexity is shifted to the sender under (A) for higher p. As R increases, there is a slow descent to an asymptotically constant value. For constant p, both $E[Y^A]$ and $E[Y^{N2}]$ are $O(\ln R)$. By comparing curves across Figures 4.2 and 4.5 it becomes clear that under (N2), in relative terms, a greater burden is placed on the receiver than under (N1) even for lower loss probabilities. This is as desired as we wish to shift processing complexity on to the receivers.

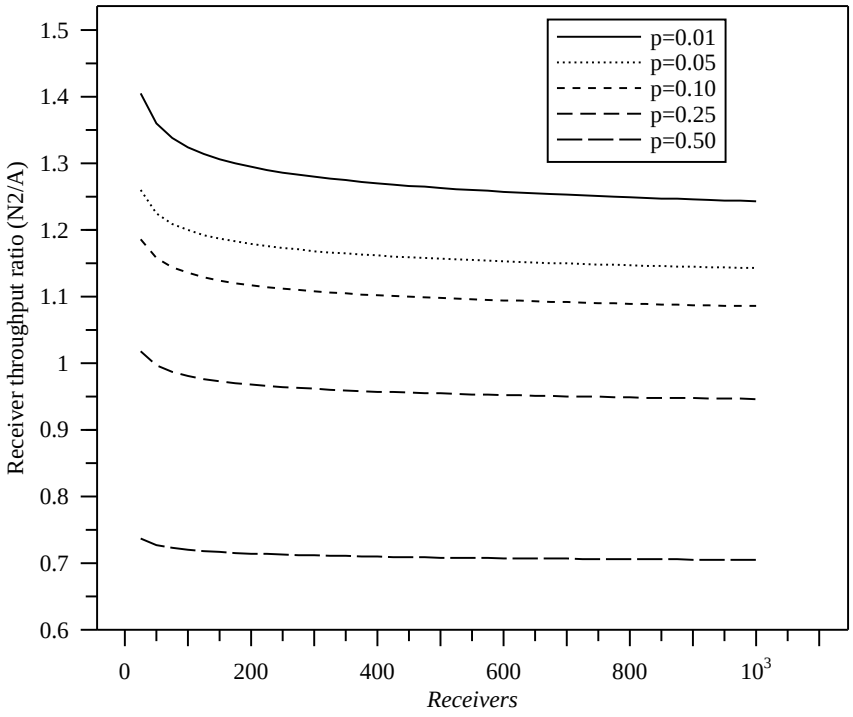

Figure 4.5. Ratio of receiver throughputs for (N2) and (A): $\Lambda_r^{N2}/\Lambda_r^A$ vs R

Figures 4.6, 4.7 and 4.8 show the absolute values of throughput at the sender and receiver for (A), (N1) and (N2) respectively. For all the graphs, the curves for $p = 0.05$ and $p = 0.25$ are shown. As can be seen from Figure 4.6, for both values of p, the sender is the bottleneck under protocol (A). As the number of receivers increases, the sender has to either process more acknowledgments (for low loss) or more timeouts (for high loss). The receiver does better at both probability levels, though clearly there is higher throughput attained under the lower loss probability. Figures 4.7 and

4.8 show that the sender remains the bottleneck for overall throughput for (N1) and (N2) as well. However, the gap between the performance at sender and receiver is narrower for (N1) than (A), and still narrower for (N2) over (N1). This is especially clear for (N2) for low numbers of receivers when the sender and receiver throughputs are very close to each other.

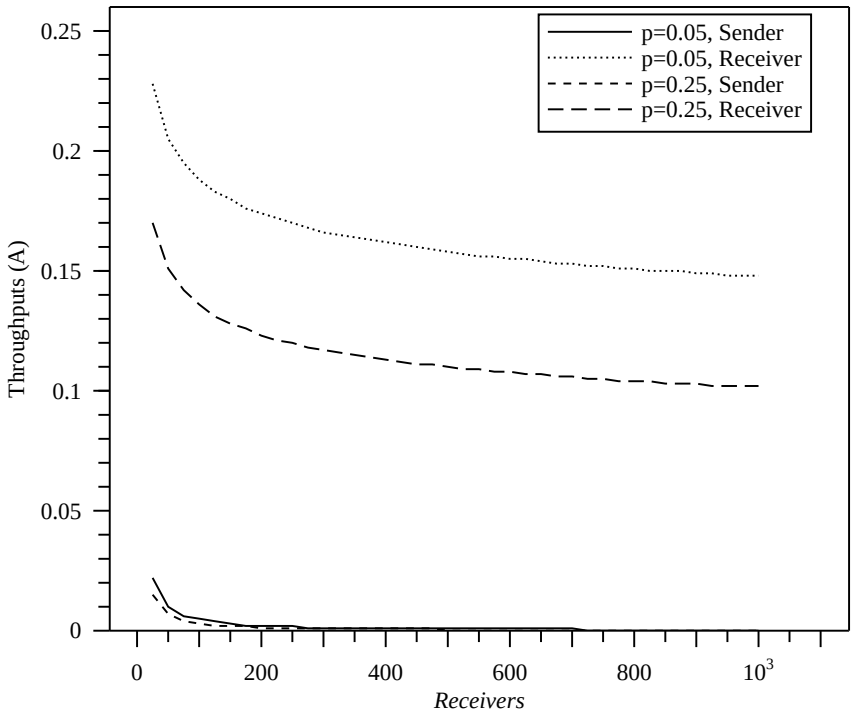

Figure 4.6. Sender and receiver throughputs for (A): Λ_s^A, Λ_r^A vs R

Finally, an interesting observation can be made from Figure 4.9. Here we plot the number of supportable receivers for different values of processor speed under the three protocols of (A), (N1) and (N2) for $p = 0.01$. Figure 4.9 is obtained by setting to 1 the speed of a processor that can support exactly one receiver under protocol (A) when $E[X_p] = E[X_t] = \ldots = 1$ as for all the earlier figures. In other words, if $\mu^\omega[R]$, $\omega \in \{A, N1, N2\}$ is the speed of a processor that can support at most R recievers under protocol ω, we set $\mu^A[1] = 1$. Since $E[M]|_{R=1} = 1/(1-p)$, we can write

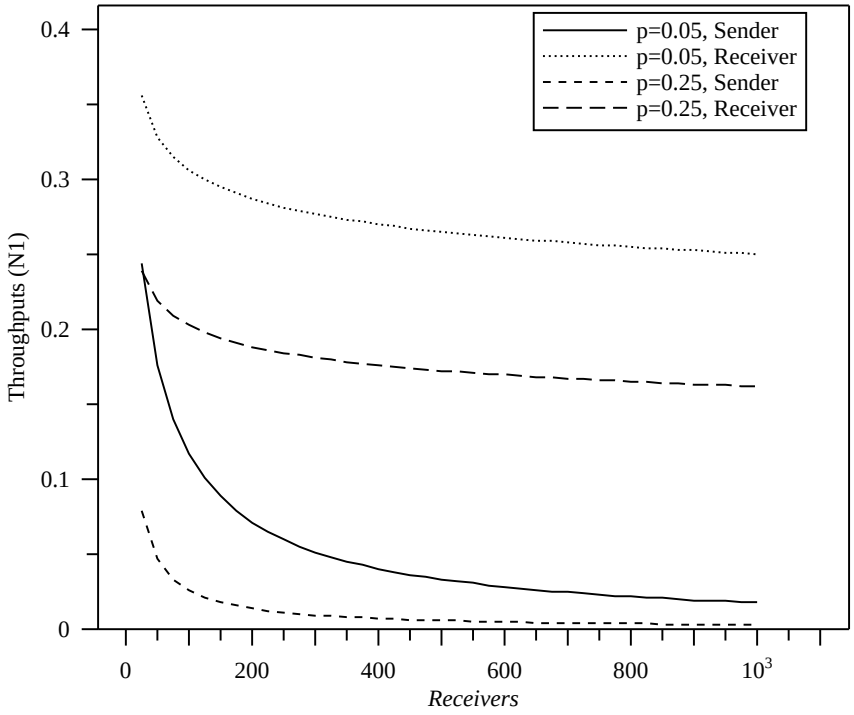

Figure 4.7. Sender and receiver throughputs for (N1): $\Lambda_s^{N1}, \Lambda_r^{N1}$ vs R

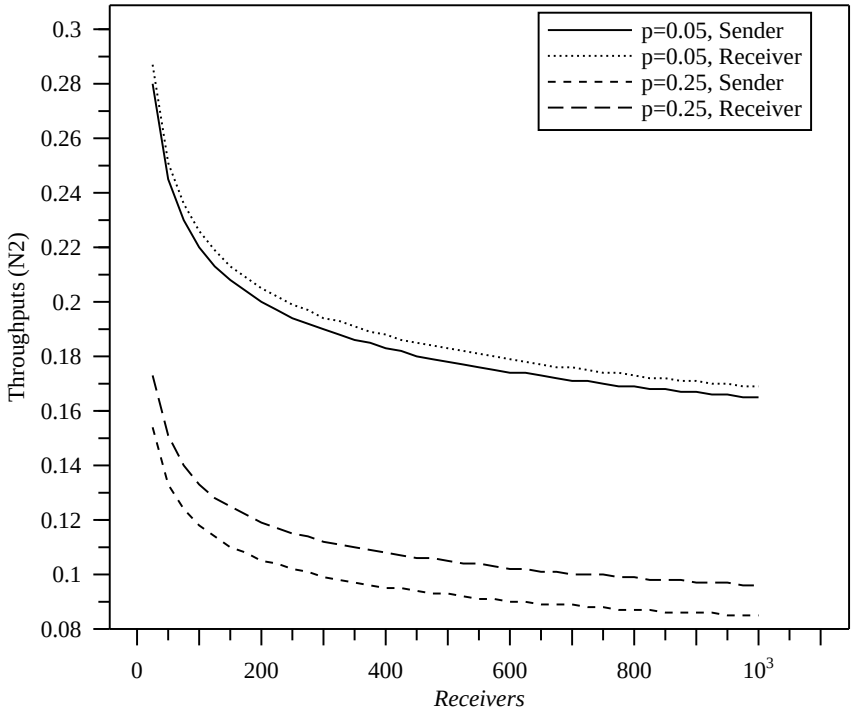

Figure 4.8. Sender and receiver throughputs for (N2): $\Lambda_s^{N2}, \Lambda_r^{N2}$ vs R

$$E[X^A]|_{R=1,p=0.01} = \frac{1}{\mu^A[1]}\frac{3-p}{1-p} = 3.02.$$

Using such a processor as the baseline, we can readily write using Equations (4.4), (4.11) and (4.12)

$$\mu^A[R] = 0.33 \times E[M](2 + R \times 0.99),$$

$$\mu^{N1}[R] = 0.33 \times (1 + E[M] + R/99)$$

and

$$\mu^{N2}[R] = 0.33 \times (2E[M]).$$

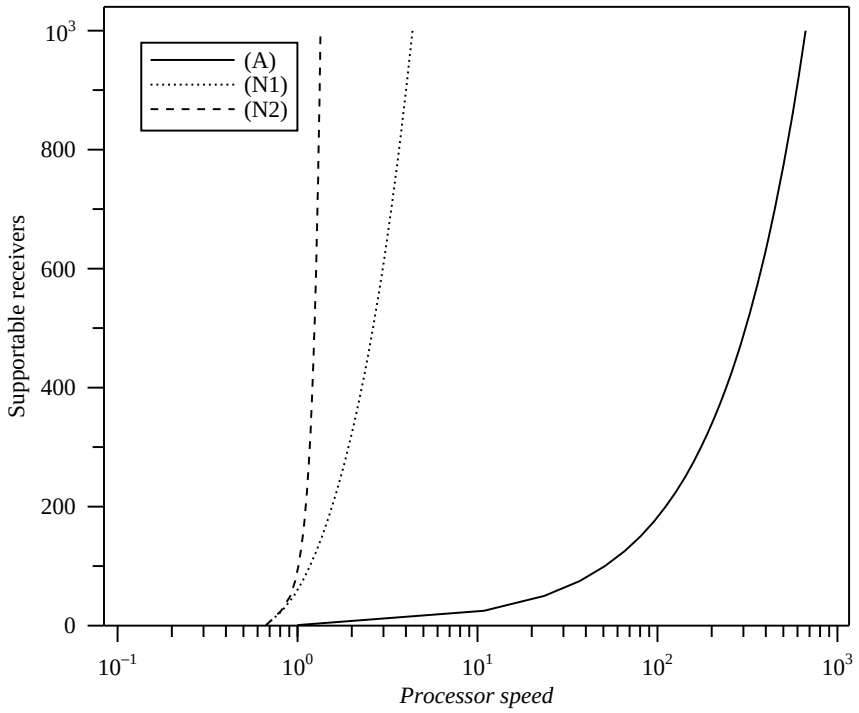

Figure 4.9. Supportable receivers vs. processor speed under (A), (N1), (N2)

The plots of Figure 4.9 are obtained from the above expressions and by treating processor speed as the independent variable and supportable receivers as the dependent variable. As can be seen from the figure, both (N1) and (N2) can support far more receivers than (A) for a much smaller processor speed. The number of supportable receivers rises sharply under (N1) and (N2) for only marginal increases in the

processor speed, whereas there has to be a substantial increase in the processor speed for there to be any noticeable increase in the number of supportable receivers under (A). Protocol (N2) demonstrates better performance than (N1) for large numbers of receivers.

4.6.2 Unequal Processing Costs for Components of Host Protocol

We now consider a different workload in which we set the cost of sending or receiving a packet to be twice that of all the other processing costs. In other words, we set $E[X_p] = E[Y_p] = 2$ and $E[X_t] = E[X_a] = E[X_n] = E[X_f] = E[Y_t] = E[Y_a] = E[Y_n] = E[Y_f] = 1$. These are plausible normalized values in some machines. For example, in the SGI Challenge (100Mhz, R4400 multiprocessor), receive time for a 10 byte packet is 44.69 μsecs while context switch time (which is most of the processing associated with all other components of the host protocol) is 21.95 μsecs.

The performance of the three protocols for this workload is very similar to that seen in the previous subsection where we had equal processing costs for all the components of protocol processing being equal. The system throughput ratios for (N1) and (A) are virtually the same as shown in Figure 4.1. However, when (N2) and (A) or (N2) and (N1) are compared, there is a lowering of the system throughput ratio to reflect the greater amount of processing necessary for the successful receipt of a packet as shown in figures 4.10 and 4.11. This is because the percent drop in sender throughput is higher for (N2) than for (N1) or (A) when we increase X_p relative to the other costs. We can see from Equation 4.12 that it is the term involving $E[X_p]$ that dominates the sender throughput expression for (N2). This is not the case for (N1) and (A). However, the throughput ratios that we observe are still in same the orders of magnitude as can be seen by comparing Figure 4.10 with Figure 4.3 and Figure 4.11 with Figure 4.4. Thus, the conclusions that we have drawn on the superiority of (N2) still hold.

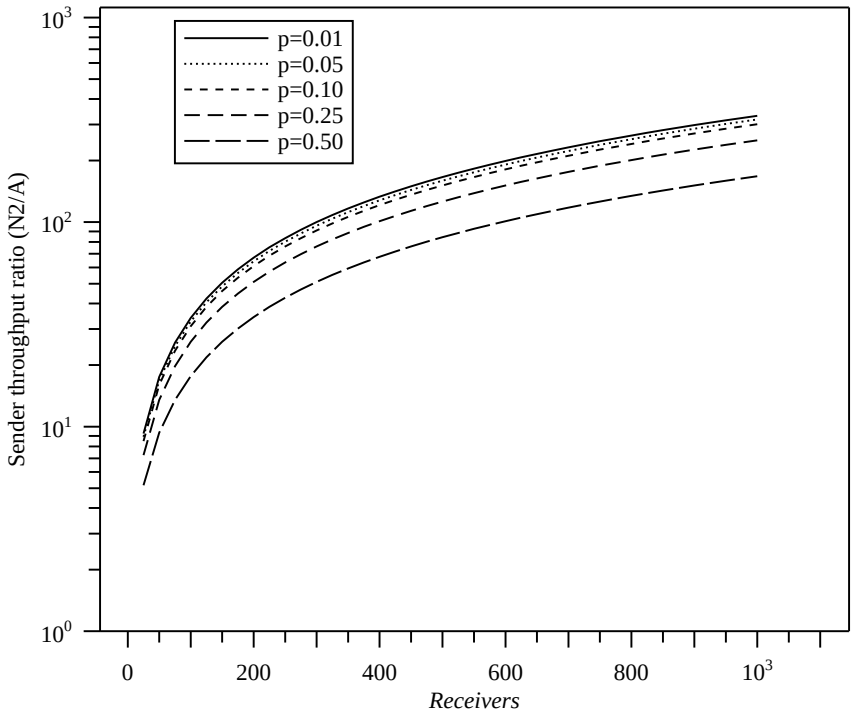

Figure 4.10. Unequal costs: $\Lambda_r^{N2}/\Lambda_r^A$ vs R

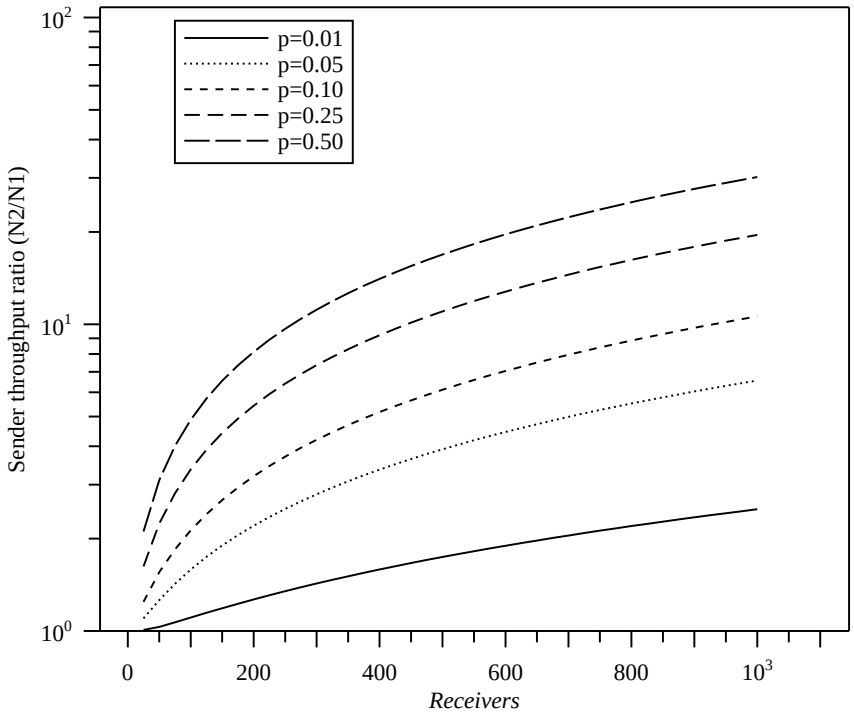

Figure 4.11. Unequal costs: $\Lambda_s^{N2}/\Lambda_s^{N1}$ vs R

4.7 Conclusions

In this chapter, we examined examined the problem of providing reliable multicast communication in large scale networks, when possibly thousands of hosts may participate in a multicast group. We studied three generic error control protocols, one that was sender-initiated (ACK-based) and two that were receiver-initiated (NAK-based). An important feature of our analysis of these protocols is that we focused on *host processing* as the constrained resource; previous analyses of reliable multicast protocols have focussed on network bandwidth as the scarce resource– a resource that we believe is becoming of less importance as network bandwidths continue to increase and multicasting emerges as an important communication paradigm. Our analyses provide a quantitative demonstration of the superiority of receiver-initiated approaches over sender-initiated approaches.

In our analyses, we assume independent loss events at the various receivers. While we expect this assumption to give us pessimistic bounds on the throughputs, whether relaxing this assumption affects the throughput expressions for the three protocols studied in different ways is a question open to investigation.

Another broad arena for future investigation is the scalability performance of optimized protocols. For example, the possibility of using local groups for error recovery rather than approaching the global sender for retransmissions is matter that remains to be studied. Various optimizations of sender-initiated schemes– such as the grouping of positive acknowledgments– are possible and can be analyzed in a manner analogous to the one given in this chapter.

So far, we have restricted our attention to throughput performance of the three protocols. We conjecture that for high loads and large numbers of receivers, the relative delay performance of the three protocols will be similar to the relative throughput performance. It is likely that for low numbers of receivers the delay performance of (N1) will be better than that for (N2), but this is a matter that remains to be investigated in a precise quantitative way. If the applications need to meet strict

time constraints, full reliability may not be possible. An interesting issue for study then would be semi-reliability and the design and analysis of protocols for delay loss tolerant applications.

CHAPTER 5

SUMMARY AND FUTURE WORK

5.1 Summary of the Dissertation

Here we summarize the research presented in this dissertation. In Chapter 1 we described the different requirements of multimedia applications and discussed the need for allocation of resources in the environments that support these applications. We introduced three specific resource-allocation problems and explained the motivation for studying each of these problems. The three areas that we identified for study were link allocation within the network, host-processor allocation at the edge of the network, and protocols for multicast applications when host processing power is the constrained resource. We indicated the need to study the performance of real-time scheduling algorithms and the scalability of multicast protocols in these three areas. In Chapters 2, 3 and 4 we discussed each of these problems in detail.

In Chapter 2 we looked at the link allocation problem by studying the issue of trading off the performance of two classes of real-time traffic with different packet-scheduling algorithms that can be implemented at an output multiplexer of a switching node. Three algorithms were studied; two of these– priority scheduling and minimum laxity thresholding– are traditional schemes. We defined a new "balancing" scheduling discipline as the third scheme studied. Both the balancing scheme and MLT are parameterized disciplines that enable us to trade the performance of one class against that of another. For the three scheduling algorithms, we obtained the delay loss performance of two classes of traffic which have identical local (nodal) deadlines, both through analysis and by simulation for different traffic arrival patterns. We demonstrated how, for a geometric bulk arrival process, in certain operating regions

the new balancing scheme provides better performance for *both* classes of traffic than the other two schemes. For a superposed on/off voice source traffic model, and also in other operating regions for the geometric bulk arrival process, we showed how strict priority scheduling is a reasonable option. We concluded that when delay loss probabilities are low, complex link-level scheduling algorithms are not essential.

In Chapter 3 we examined the processor allocation problem by studying how different CPU scheduling algorithms compare with regard to the number of preemptions induced by different dynamic and static algorithms for periodic task-sets. We showed through a sample path proof that there are always more preemptions under rate monotonic scheduling than under earliest deadline scheduling. We obtained through simulations the actual distributions of this difference in the number of preemptions between RM and ED for well-known task-sets. We showed how there is a high probability of there being significant difference (greater than 4.5% for the avionics task-set) in the actual numbers of preemptions under the two disciplines. We also showed how scaling the processor speed up or down either decreases or increases both the average and the maximum difference in the number of preemptions under RM and ED. We also studied via simulations how the structure of the task-set can effect the relative performance of the scheduling policies. If the ratio of the highest period in the task-set to the smallest period is very large, the sample path behavior is dominated by the smallest period jobs under both ED and RM. Consequently there is little relative difference in the performance of the two algorithms. Another insight relates to task-sets which have some tasks whose periods are very close to one another. Here, there could always be an initial phasing under RM such that there are repeated premptions of jobs of a given period by jobs whose periods are only slightly smaller. We surmise this last result by studying the performance of an algorithm called $HEHP$, which is a slight variation of RM, and which demonstrates a maximum difference of over 188% in the number of preemptions relative to ED. We also simulated a processor through the modeling of some operating system overheads,

and demonstrated how there is a slight increase in the number of preemptions under both ED and RM as compared to when these overheads are not taken into account.

In Chapter 4, we studied the performance of three different reliable multicast protocols when host processing power rather than network bandwidth is the bottleneck. One of these protocols was a sender-initiated protocol– i.e., the responsibility for initiating recovery from errors was on the sender. The other two were receiver-initiated protocols where the onus for error-recovery lay on the receivers. For each of these three protocols we obtained sender and receiver throughput expressions by explicitly modeling various components of host protocol processing. Using these throughput expressions, we obtained complexity results for each protocol showing how the processing requirements imposed on the hosts change as the number of receivers is scaled upwards. These complexity results clearly show the superiority of receiver-initiated protocols as compared to sender-initiated ones with regard to scalability. We also obtained throughput ratios across these protocols as a function of the number of receivers for different workloads. These numerical results demonstrate that there can be orders of magnitude difference in performance between receiver-initiated protocols and sender-initiated protocols.

The essential conclusion that we may draw from all the work described in this dissertation is that the manner in which different resources are used in multimedia applications has to be considered carefully. This can lead to more efficient scheduling algorithms and more scalable protocols that improve the overall quality of service provided to the users of the applications.

5.2 Suggestions for Future Work

In this section we indicate a few of the many resource-allocation problems that need to be addressed for multimedia applications.

In the study of link scheduling algorithms, an open issue is their performance when the different arrival streams to an output multiplexer are themselves correlated. In

this dissertation, we modeled arrival processes that demonstrated correlation from time slot to time slot, but the different traffic classes themselves were mutually independent. This assumption could be relaxed and, in addition, other traffic models such as those for video source be used.

The performance of preemptive processor scheduling algorithms with regard to numbers of preemptions in the context of a comprehensive model of operating system overheads remains to be examined. In our simulation work we model only a few of these costs. For example, actual implementation of real-item operating systems like Mach include timer costs which are not captured in our simulations. Further, an interesting possibility to explore is whether an order relationship can be estabished between the numbers of preemptions for different disciplines by means of an analytical proof when these overheads are taken into account.

In our study of scalable multicast protocols, we confined our attention to fully reliable protocols. These protocols are realistic in lossy environments only when no real-time constraints are imposed on data transfer. This suggests that a delay analysis of the protocols studied would provide useful information in the context of multimedia applications. Further, the study of different protocols when fully-reliable service is not demanded is an interesting open issue. Other optimizations of fully reliable multicast protocols can also be studied– e.g., those that rely on neighbors for error-recovery rather than the global sender.

In addition to all of the above extensions to the work in this dissertation, there is also the issue of synchronized media presentation in multimedia workstations. The resource bottlenecks in this context can be identified and policies arrived at to properly allocate them.

APPENDIX A

LINK SCHEDULING: TRANSITION MATRICES

In this appendix we indicate how the transition probability matrix may be obtained for a 2-D Markov chain model of an output multiplexer at a switching node. There are two classes of traffic.

Let g_1 and g_2 be parameters of geometric processes that represent the probability of succesful packet arrival for Class 1 and Class 2 respectively. In order to explain how the entries for the transition matrix are obtained, we first define several quantities:

- $\alpha_i = g_1^i(1 - g_1)$ - probability of having i Class 1 arrivals in a slot
- $\beta_i = g_2^i(i - g_2)$ - probability of having i Class 2 arrivals in a slot
- $\alpha_{1+} = g_1$ - probability of at least one Class 1 arrival in a slot
- $\beta_{1+} = g_2$ - probability of at least one Class 2 arrival in a slot
- $p_i = (1 - g_1)^i g_1$ - probability that the laxity of the next-to-last oldest queued Class 1 packet is i slots greater than the laxity of the oldest queued Class 1 packet
- $s_i = (1 - g_2)^i g_2$ - probability that the laxity of the next-to-last oldest queued Class 2 packet is i slots greater than the laxity of the oldest queued Class 2 packet
- $q_i = (1 - \sum_{j=0}^{i} p_j)$ - probability that there are no Class 1 arrivals up to i slots after the arrival of the present minimum laxity Class 1 packet
- $z_i = (1 - \sum_{j=0}^{i} s_j)$ - probability that there are no Class 2 arrivals up to i slots after the arrival of the present minimum laxity Class 2 packet.

Using all of the above, we now consider the priority case in detail. MLT and balancing schemes can be understood in an analogous fashion. Let the deadlines of Class 1 and Class 2 packets be r_1 and r_2 respectively. Then, the overall transition matrix is of dimension $((r_1+1)(r_2+1)) \times ((r_1+1)(r_2+1))$. This is actually a block partitioned matrix with each block being of dimension $(r_1+1) \times (r_1+1)$. Each of these blocks represents the transitions from an initial Class 1 minimum laxity of x_1' to a final Class 1 minimum laxity of x_1''. The location of the block $(r_1+1) \times (r_1+1)$ within the larger matrix gives the values of x_2' and x_2'' which are the initial and final Class 2 minimum laxities respectively. The basic form of the transition matrix, P^{pri} for the priority case is as follows:

$$P^{pri} = \begin{bmatrix} A_0 & A_1 & A_2 & \cdots & A_{r_2-2} & A_{r_2-1} & Z_{r_2-1} \\ B_0 & B_1 & B_2 & \cdots & B_{r_2-2} & B_{r_2-1} & Z'_{r_2-2} \\ 0 & B_0 & B_1 & \cdots & B_{r_2-3} & B_{r_2-2} & Z'_{r_2-3} \\ 0 & 0 & B_0 & \cdots & B_{r_2-4} & B_{r_2-3} & Z'_{r_2-4} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & B_0 & B_1 & Z'_1 \\ 0 & 0 & 0 & \cdots & 0 & C_{1+} & C_0 \end{bmatrix}$$

Each entry in P^{pri} is an $(r_1+1) \times (r_1+1)$ matrix. Each of the zero entries in P^{pri} above corresponds to a matrix of all zeros. The rows of P^{pri} itself correspond to values of $x_2' = 1, 2, \ldots, r_2, 0$ and the columns to $x_2'' = 1, 2, \ldots, r_2, 0$. Note that the minimum laxity value of 0 corresponds to the *last* row and *last* column and denotes the state of there being no packets of the corresponding class in the system. We use this sequence of values for minimum laxities here because we wish to highlight the empty queue as a special case and separate it from the other values of minimum laxities. The structure of the transition matrix is better understood in this way as, in physical reality, when there is a arrival to an empty queue, the minimum laxity jumps to the value of the deadline.

We give each of the entries of P^{pri} in a simple way. We define a basic matrix, M that gives the transition probabilities for Class 1 minimum laxities alone:

$$M = \begin{bmatrix} p_0 & p_1 & p_2 & \cdots & p_{r_1-2} & p_{r_1-1} & q_{r_1-1} \\ p_0 & p_1 & p_2 & \cdots & p_{r_1-2} & p_{r_1-1} & q_{r_1-1} \\ 0 & p_0 & p_1 & \cdots & p_{r_1-3} & p_{r_1-2} & q_{r_1-2} \\ 0 & 0 & p_0 & \cdots & p_{r_1-4} & p_{r_1-3} & q_2 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & p_0 & p_1 & q_1 \\ 0 & 0 & 0 & \cdots & 0 & \alpha_{1+} & \alpha_0 \end{bmatrix} \tag{A.1}$$

Let $D_i = \mathbf{diag}(s_i, s_i, \ldots, s_i)$, $i = 0, 1, \ldots, r_2 - 1$ be a diagonal matrix of dimension $(r_1+1)\times(r_1+1)$ whose diagonal elements are all equal to s_i. The first (r_2-1) columns of the first row of P^{pri} are given by

$$A_i = D_i M, \ i = 0, 1, \ldots, r_2 - 1.$$

We can write all the other entries of P^{pri} in a similar manner as shown below. In each case, the diagonal matrices are of dimension $(r_1 + 1) \times (r_1 + 1)$.

$$\begin{aligned} Z_{r_2-1} &= \mathbf{diag}(z_1, z_1, \ldots, z_1)M. \\ Z'_{r_2-i} &= \mathbf{diag}(0, 0, \ldots, z_i)M, \ i = 2, 3, \ldots, r_2 - 1. \\ B_0 &= \mathbf{diag}(1, 1, \ldots, s_0)M. \\ B_i &= \mathbf{diag}(0, 0, \ldots, s_i)M, \ i = 1, 2, \ldots, r_2 - 1. \\ C_{1+} &= \mathbf{diag}(\beta_{1+}, \beta_{1+}, \ldots, \beta_{1+})M. \\ C_0 &= \mathbf{diag}(\beta_0, \beta_0, \ldots, \beta_0)M. \end{aligned}$$

Let $P^{pri}_{i,j}$ represent the matrix entry at the ith block row, and the jth block column of P^{pri}. To intuitively understand why the form of P^{pri} is as shown above, we first look at the zero entries of P^{pri}. For example, $P^{pri}_{4,2}$ is a zero matrix. This entry is for state transitions from $(x'_1, 4)$ to $(x''_1, 2)$ with x'_1 and x''_1 taking values $1, 2, \ldots, r_1, 0$. Since the state indicates the laxity of the minimum laxity packet for each class, we see that it is *impossible* for the minimum laxity of Class 2 packets to drop from 4 to 2 in one slot; if a Class 1 packet is served, the minimum Class 2 laxity merely goes from 4 to 3. If the Class 2 packet is served (in the priority case, this can only happen

if there are no Class 1 packets), the new minimum laxity Class 2 packet could only have arrived in the same slot as the Class 2 packet just served, or later. If it arrived in the same slot, the new Class 2 minimum laxity would be 3; if it arrived later, it would be greater than 3. Thus the probability of going from $(x_1^{'}, 4)$ to $(x_1^{''}, 2)$ is zero.

We now turn our attention to one of the other entries in the transition matrix, say $P_{3,2}^{pri} = B_0$. The form of B_0 is as follows:

$$B_0 = \begin{bmatrix} p_0 & p_1 & p_2 & \cdots & p_{r_1-2} & p_{r_1-1} & q_{r_1-1} \\ p_0 & p_1 & p_2 & \cdots & p_{r_1-2} & p_{r_1-1} & q_{r_1-1} \\ 0 & p_0 & p_1 & \cdots & p_{r_1-3} & p_{r_1-2} & q_{r_1-2} \\ 0 & 0 & p_0 & \cdots & p_{r_1-4} & p_{r_1-3} & q_2 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & p_0 & p_1 & q_1 \\ 0 & 0 & 0 & \cdots & 0 & s_0\alpha_{1+} & s_0\alpha_0 \end{bmatrix}$$

The first r_1 rows of B_0 correspond to transitions $(x_1^{'}, 3)$ to $(x_1^{''}, 2)$ where $x_1^{'} = 1, 2, \ldots, r_1$. Since this is the priority discipline, the transition from 3 to 2 for x_2 happens automatically when there exists a Class 1 packet to be served. The probability of going from 2 to 1 for x_1 is p_0. This is because the current minimum laxity Class 1 packet is served and for the next minimum laxity Class 1 packet to have a laxity of 1, it should have arrived in the same slot as the packet that is currently served. The first row is the same as the second row because of the memoryless property of the geometric distribution. Similarly, we can see that all the other entries for the first r_1 rows are correct. The last row corresponds to transitions from $(0, 3)$ to $(x_1^{''}, 2)$. Here, it is the Class 2 packet that is served and the probability of x_2 going from 3 to 2 is s_0. From an initial value of 0, the value of x_1 can either go to r_1 (the Class 1 deadline) or remain at 0. The first of these corresponds to the case of there being at least one Class 1 arrival in the current slot and the second to the case of there being no Class 1 arrivals in the current slot.

The basic form of the transition matrix for the MLT case, P^{mlt} is exactly the same as that for the priority case. By examining P^{pri}, we see that there are entries that are repeated. A similar repetitive structure is obtained for P^{mlt}, though the

actual entries in the component matrices of P^{mlt} differ from from those of P^{pri}. The rows of P^{mlt} and P^{pri} corresponding to $x_2' = 0$ are the same, i.e., the submatrices C_{1+} and C_0 are the same in the two disciplines. The other submatrices in P^{mlt} different from the those in P^{pri}.

We define a basic submatrix, M' in order to write all the other submatrices of P^{mlt}. M' is an $(r_1+1)\times(r_1+1)$ matrix that takes the following form for a threshold value of T:

$$M'_{kl} = \begin{cases} M_{kl}, & 1 \le k \le T,\ 1 \le l \le r_1+1 \\ 1, & T \le k \le r_1,\ l = k-1 \\ 0, & T \le k \le r_1,\ l \ne k-1 \\ M_{kl}, & k = r_1+1,\ 1 \le l \le r_1+1. \end{cases} \tag{A.2}$$

where M is the matrix whose form is shown in (A.1).

We define the notation $\mathbf{Diag}(n,b,c)$ to be a special form of a diagonal matrix whose first n diagonal elements take value b and whose remaining diagonal elements take value c. All the matrices below are of dimension $(r_1+1)\times(r_1+1)$ and we write the submatrices of P^{mlt} as

$$\begin{aligned} A_i &= \mathbf{diag}(s_i, s_i, \ldots, s_i)M',\ i = 0,1,\ldots,r_2-1. \\ Z_{r_2-1} &= \mathbf{diag}(z_1, z_1, \ldots, z_1)M'. \\ Z'_{r_2-i} &= \mathbf{Diag}(T, 0, z_i)M',\ i = 2,3,\ldots,r_2-1. \\ B_0 &= \mathbf{Diag}(T, 1, s_0)M'. \\ B_i &= \mathbf{Diag}(T, 0, s_i)M',\ i = 1,2,\ldots,r_2-1. \end{aligned}$$

For example, $P^{mlt}_{3,2} = B_0$ is written for $T = 3$ as follows:

$$P^{mlt}_{3,2} = \begin{bmatrix} p_0 & p_1 & p_2 & \cdots & p_{r_1-2} & p_{r_1-1} & q_{r_1-1} \\ p_0 & p_1 & p_2 & \cdots & p_{r_1-2} & p_{r_1-1} & q_{r_1-1} \\ 0 & p_0 & p_1 & \cdots & p_{r_1-3} & p_{r_1-2} & q_{r_1-2} \\ 0 & 0 & s_0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & s_0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & s_0\alpha_{1+} & s_0\alpha_0 \end{bmatrix}$$

It is instructive to compare $P^{mlt}_{3,2}$ and $P^{pri}_{3,2}$. It can be seen that the first three rows of the two matrices are the same. This is because up to the value of the threshold,

priority scheduling and MLT behave in the same fashion. The last row is also the same because that corresponds to $x_1' = 0$, i.e., there are no Class 1 packets that may be served in this slot. Rows 4 through r_1 are different in accordance with the manner in which MLT differs from the priority discipline.

The form of the transition matrix for the balancing discipline is a little more difficult to obtain. Since it is the *difference* between x_2' and x_1' that determines which class of traffic is served, care must be taken while writing the transition matrix. The nature of the matrix is given below in terms of the entries of P^{pri} and P^{mlt}. In understanding the following, it is useful to remember that the row index of P^{bal} is the same as x_2' and the column index is the same as x_2'' except for the last row and last column which correspond to $x_2' = 0$ and $x_2'' = 0$ respectively.

Let $P^{mlt}(T)$ denote the transition matrix in the MLT case for threshold of T, and if $T \geq r_1$, the Class 1 deadline, let $P^{mlt}(T) = P^{pri}$. Further, let B be the parameter of the balancing discipline. Then, if $P^{bal}_{i,j}$ denotes the submatrix at the ith block row and jth block column of P^{bal}, we may write:

$$\begin{aligned} P^{bal}_{i,j} &= P^{mlt}_{i,j}(i+B-1),\ 1 \leq i \leq r_2,\ 1 \leq j \leq r_2+1 \\ P^{bal}_{(r_2+1),j} &= P^{pri}_{(r_2+1),j},\ 1 \leq j \leq r_2+1 \end{aligned}$$

The reason for the above form is that if $x_2' = i$, Class 1 traffic is served so long as $x_1' \leq i + B - 1$. This is the same as the behavior of the MLT discipline when a threshold of $i + B - 1$ is imposed on the minimum laxity of Class 1 traffic. If $i + B - 1 \geq r_1$, by our notation, we have the balancing scheme behaving exactly as the priority scheme. This is so because Class 2 minimum laxity never gets to be at least B less than the Class 1 minimum laxity which can only take values up to r_1. Further, in the above form of P^{bal}, we can see that when there are no Class 2 packets ($x_2' = r_2 + 1$), the balancing scheme is the same as the priority scheme.

APPENDIX B

MULTICAST PROTOCOLS: COMPLEXITY OF E[M]

In this appendix we derive the complexity of the expected number of transmissions from a sender to R receivers needed for all the receivers to correctly receive the packet. The probability that any particular receiver will not receive the packet during any transmission is p and is independent of the loss probability at all other receivers.

Define an i.i.d. sequence of r.v.'s $\{M_r\}_1^R$ such that

$$P[M_r \geq x] = \begin{cases} p^{\lfloor x \rfloor - 1}, & x > 1 \\ 1, & 0 \leq x \leq 1. \end{cases} \tag{B.1}$$

Notice that for integer m, $P[M_r \geq m] = p^{m-1}$, which is the same behavior as that of a geometric random variable of success parameter $(1-p)$. This geometric random variable has the interpretation of being the number of transmissions necessary for successful receipt of the packet at receiver r.

Let $M = \max_r \{M_r\}$. If each of the M_r's was replaced by the corresponding geometric r.v., M would have the interpretation of being the number of transmissions necessary for a packet to be correctly received at all receivers.

Let N_r be an exponentially distributed random variable with parameter λ that satisfies the following:

$$M_r - 1 \leq_{st} N_r.$$

Hence for integer m, we can write

$$P[M_r - 1 \geq m] \leq P[N_r \geq m].$$

Or,

$$p^m \leq e^{-\lambda m}.$$

Therefore

$$\lambda \leq -\ln p.$$

If we set $\lambda = -\ln p$ we get

$$P[N_r \geq x] = p^x \leq \begin{cases} p^{\lfloor x \rfloor - 1}, & x > 1 \\ 1, & 0 \leq x \leq 1. \end{cases}$$

The RHS of the inequality above is simply the behavior of M_r as shown in Equation B.1. Therefore $P[N_r \geq x] \leq P[M_r \geq x]$. Hence we can write the following:

$$M_r - 1 \leq_{st} N_r \leq_{st} M_r.$$

Adding 1 to the first two terms of the above order relation and combining it with the relation of the last two terms above, gives us the following order relation

$$N_r \leq_{st} M_r \leq_{st} N_r + 1.$$

Hence

$$\max_r \{N_r\} \leq_{st} \max_r \{M_r\} \leq_{st} 1 + \max_r \{N_r\}.$$

Replacing $\max_r\{M_r\}$ in the above by M and taking expectations we get

$$E[N] \leq E[M] \leq 1 + E[N]$$

where N is a random variable that is the maximum of R exponentially distributed random variables each of parameter $-\ln p$.

It is well known that the mean of the maximum of R independent exponential random variables is equal to a harmonic series divided by the parameter of the exponential random variables. In our context we can therefore write

$$\frac{H_R}{-\ln p} \leq E[M] \leq 1 + \frac{H_R}{-\ln p}$$

where $H_R = \sum_{i=1}^{R} 1/i$. Hence, $E[M] = H_R/(-\ln p) + O(1)$. It is a well known result that H_R is $O(\ln R)$. Thus, we can also express this as $E[M] = O(1 + \ln R/(-\ln p))$.

Bibliography

[1] CCITT Study Group XVIII, Report R 34, I.211 B-ISDN service aspects (Draft Recommendation), June 1990.

[2] Schulzrinne, H., "Voice Communication Across the Internet: A Network Voice Terminal", Reseach Report, Dept. of C.S., UMass, Amherst, July 1992.

[3] Frederick, R., "nv", Manual Pages, Xerox Palo Alto Research Center.

[4] Jacobson, V., and McCanne, S., "Using the LBL Network 'Whiteboard"', Lawrence Berkeley Laboratory, Berkeley, CA.

[5] Nagarajan, R., Kurose, J., and Towsley, D., "Local allocation of End-to-End Quality-of-Service in High-Speed Networks". In *IFIP Workshop on Modeling and Performance Evaluation of ATM Technology*, Le Martinique, January 1993.

[6] Ferrandiz, J., and Lazar, A., "Admission Control for Real-Time Packet Sessions", *Proceedings of IEEE Infocom '91*, pp. 553–559, Bal Harbour, Florida, April 1991.

[7] Karlsson, G., and Vetterli, M., "Packet Video and Its Integration into the Network Architecture", *IEEE, Journal on Selected Areas in Communications*, vol. 7, pp. 739-751, June 1989.

[8] Ghanbari, M., "Two-Layer Coding of Video Signals for VBR Networks", *IEEE, Journal on Selected Areas in Communications*, vol. 7, pp. 771-789, June 1989.

[9] Chipalkatti, R., Kurose, J., and Towsley, D., "Scheduling Policies for Real-Time and Non-Real-Time Traffic in a Statistical Multiplexer", *Proceedings of IEEE Infocom '89*, Ottawa, April 1989.

[10] Golestani, S. J., "A Framing Strategy for Congestion Management", *IEEE Journal on Selected Areas in Communications,* vol. 7, pp. 1064–1077, September 1991.

[11] Golestani, S. J., "Congestion-Free Commmunication in High-Speed Packet Networks", *IEEE Transactions on Communications,* vol. 39, pp. 1802–1812, December 1991.

[12] Liu, C.L., and Layland, J.W., "Scheduling Algorithms for Multiprogramming in a Real-Time Environment", *Journal of the ACM*, vol. 20, pp. 46–61, 1973.

[13] Strosnider, J., Lehoczky, J., Sha, L., and Tokuda, H., "Fixed Priority Scheduling Theory for Hard Real-Time Systems". In A.M. van Tilborg and G.M. Koob, editors, *Foundations of Real-Time Computing*, Chapter 1, pp. 1–30. Kluwer Academic Publishers, 1991.

[14] Arakawa, H., Katcher, D., and Strosnider, J., "Engineering and Analysis of Fixed Priority Schedulers". To appear in *IEEE Transactions on Software Engineering.*

[15] Strosnider, J., Arakawa, H., Katcher, D., and Tokuda, H., "Modeling and Validation of the Real-Time Mach Scheduler", *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Santa Clara, May 1993.

[16] Pingali, S., Kurose, J., and Towsley, D., "On Comparing the Number of Preemptions under ED and RM Scheduling algorithms". Submitted to *IEEE Transactions on Computers.*

[17] Jacobson, V., and McCanne, S., "vat", Manual Pages, Lawrence Berkeley Laboratory, Berkeley, CA.

[18] Padmanabhan, L., "Design and Implementation of a Shared White-Board", M.S. Project, Dept. of Computer Science, UMass, Amherst, MA 01003, May 1993.

[19] Casner, S., "First IETF Internet Audiocast", *ACM SIGCOMM Computer Communication Review*, vol. 22, pp. 92–97, July 1992.

[20] Crowcroft, J., and Paliwoda, K., "A Multicast Transport Protocol", *Proceedings of ACM SIGCOMM '88*, pp. 247–256, Stanford, August 1988.

[21] Yavatkar, R., and Manoj, L., "Optimistic Approaches to Large-Scale Dissemination of Multimedia Information", *Proceedings of ACM Multimedia '93*, August 1993.

[22] Panwar, S.S., Towsley, D., and Wolf, J.K., "Optimal Scheduling Policies for a Class of Queues with Customer Deadlines to the Beginning of Service", *Journal of the ACM*, vol. 35, num. 4, pp. 832–844, October 1988.

[23] Deering, S., and Cheriton, D., "Multicast Routing in Datagram Internetworks and Extended LANs", *ACM Transactions on Computer Systems*, vol. 8, pp. 85–110, May 1990.

[24] Lim, Y., and Kobza, J., "Analysis of a Delay-Dependent Priority Discipline in a Multi-Class Traffic Packet Switching Node", *Proceedings of IEEE Infocom '88*, pp. 889–898, New Orleans, March 1988.

[25] Schulzrinne, H., Kurose, J., and Towsley, D., "Congestion Control for Real-Time Traffic in High-Speed Networks," *Proceedings of IEEE Infocom '90,* pp. 543-550, San Francisco, June 1990.

[26] Verma, D., Zhang, H., and Ferrari, D., "Delay Jitter Control for Real-Time Communication in a Packet Switching Network". In *IEEE Tricomm'91*, April 1991.

[27] Ferrari, D., and Verma, D., "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, pp. 368–379, April, 1990.

[28] Clark, D., Shenker, S., and Zhang, L., "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Merchanism," *ACM SIGCOMM'92*, pp. 14–26, 1992.

[29] Schulzrinne, H., Kurose, J., and Towsley, D., "An Evaluation of Scheduling Mechanisms for Providing Best-Effort, Real-Time Communication in Wide-Area Networks", *Proceedings of IEEE Infocom '94*, Toronto, June 1994.

[30] Brady, P. T., "A Statistical Analysis of On-Off Patterns in 16 Conversations," *Bell System Technical Journal,* vol. 47, pp. 73–91, January 1968.

[31] Sriram, K., and Whitt, W., "Characterizing Superposition Arrival Processes in Packet Multiplexers for Voice and Data", *IEEE Journal on Selected Areas in Communications*, vol. 4, pp. 833–846, September 1986.

[32] Jean-Marie, A., Lefebvre-Barbaroux, S., and Chaouiya, C., "Real-Time Scheduling of Periodic Tasks". To appear in E.G. Coffman, A. Lenstra and Z. liu, editors, *Scheduling Theory and its Applications*. Wiley.

[33] Jeffay, K., Stanat, D.F., and Martel, C.U., "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks". In *IEEE Real-Time Systems Symposium*, pp 129-139, December 1991.

[34] Hanko, J.G., Kuerner, E.M., Northcutt, J.D., and Wall, G.A., "Workstation Support for Time-Critical Applications". In *2nd Intl. Workshop on Network and OS Support for Digital Audio and Video.* Heidelberg, Germany, Nov. 1991.

[35] Anderson, T.E., Bershad, B.N., Lazowska, E.D., and Levy, H.M., "Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism", *ACM Transactions on Computer Systems*, 10: 53–79, February 1992.

[36] Nain, P., and Towsley, D., "Optimal Scheduling in a Machine with Stochastic Varying Processing Rate". To appear in *IEEE Trans. on Automatic Control.*

[37] Katcher, D., and Strosnider, J., "Dynamic versus Fixed Priority Scheduling: A Case Study". Submitted to *IEEE Trans. on Software Engineering.*

[38] Bulterman, D.C.A., and van Liere, R., "Multimedia Synchronization and Unix." In *2nd Intl. Workshop on Network and OS Support for Digital Audio and Video.* Heidelberg, Germany, Nov. 1991.

[39] Chang, J., and Maxemchuck, N., "Reliable Broadcast Protocols", *ACM Transactions on Computer Systems*, vol. 2, pp. 251–273, August 1984.

[40] Towsley, D., "An Analysis of a Point-to-Multipoint Channel Using a Go-Back-N Error Control Protocol", *IEEE Transactions on Communications*, vol. 33, pp. 282–285, March 1985.

[41] Wang, J. L., and Silvester, J. A., "Optimal Adaptive ARQ Protocols for Point-to-Multipoint Communication", *Proceedings of IEEE Infocom'88*, pp. 704–713, March 1988.

[42] Gopal, I., and Jaffe, J., "Point-to-Multipoint Communication over Broadcast Links", *IEEE Transactions on Communications,* vol. 32, pp. 1034–1044, September 1984.

[43] Wang, J. L., and Silvester, J. A., "Delay Minimization of the Adaptive Go-Back-N ARQ Protocols for Point-to-Multipoint Communication", *Proceedings of IEEE Infocom'89*, pp. 584–593, April 1989.

[44] Towsley, D., and Mithal, S., "A Selective Repeat ARQ Protocol for a Point to Multipoint Channel," *Proceedings of IEEE Infocom'87*, pp. 521–526, Mar-Apr 1987.

[45] Weldon, E., "An Improved Selective-Repeat ARQ Strategy", *IEEE Transactions on Communications*, vol. 30, pp. 480–486, March 1982.

[46] Ram Chandran, S., and Lin, S., "Selective-repeat-ARQ Schemes for Broadcast Links," *IEEE Transactions on Communications,* vol. 40, pp. 12–19, January 1992.

[47] Shacham, N., and Towsley, D., "Resequencing Delay and Buffer Occupancy in Selective Repeat ARQ with Multiple Receivers," *Proceedings of IEEE Infocom'88*, pp. 515–524, March 1988.

[48] Sabnani, K., and Schwartz, M., "Multidestination Protocols for Satellite Broadcast Channels," *IEEE Transactions on Communications,* vol. 33, pp. 232–240, March 1985.

[49] Ramakrishnan, S., and Jain, B. N., "A Negative Acknowledgement with Periodic Polling Protocol for Multicast over LANs," *Proceedings of IEEE Infocom'87*, pp 502–511, Mar-Apr 1987.

[50] Jacobson, V., 1993 ARPA Networking PI Meeting, Sept. 1993.