The Preserve: Lehigh Library Digital Collections

# Application of hard real-time scheduling algorithms in periodic network transmissions.

**Citation**

Erkan, Ali. *Application of Hard Real-Time Scheduling Algorithms in Periodic Network Transmissions*. 2006, https://preserve.lehigh.edu/lehigh-scholarship/graduate-publications-theses-dissertations/theses-dissertations/application-hard.

Find more at https://preserve.lehigh.edu/

*This document is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion by an authorized administrator of Lehigh Preserve. For more information, please contact preserve@lehigh.edu.*

# Application of Hard Real-Time Scheduling Algorithms in Periodic Network Transmissions

by

Ali Erkan

Presented to the Graduate and Research Committee
of Lehigh University
in Candidacy for the Degree of
Doctor of Philosophy

in

Computer Science and Engineering

Lehigh University

January 2006

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UMI Number: 3203815

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

UMI Microform 3203815

Copyright 2006 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

December 9, 2005

Date

Dr. Terrance Boult

Dissertation Advisor

December 9, 2005

Accepted Date

Committee Members:

Dr. Donald Hillman

Dr. Terrance Boult

Dr. Drew Kessler

Dr. Michael Schulte



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# Acknowledgments

I decided to be an educator when I was watching the movie Dead Poets Society one day more than a decade ago. The task seemed simple enough: get a Ph.D. and start teaching. I never realized what a humbling experience it would be to come to a point where I would actually be typing the words of the acknowledgment of my dissertation. And as I do, I feel an overwhelming sense of gratitude towards the people who made it possible.

Selma and Metin, my dear parents... They invested so much into me. At every step of the way, they were with me even though they were physically thousands of miles away. Since the instant I was born, they loved and cared almost to the point of self-destruction. It is an honor to be their son and to carry their name.

Naomi, my wonderful wife... Her intelligence, honesty, and love have many times been the reason why I did not turn around and run away. I cannot imagine a more special soul-mate; I cannot imagine a better mother for my children.

Aylin and Michael, the "little" support system... There is nothing like the innocent face of a sleeping child to bring out the strongest determination to succeed. It was their magic that pulled me out of many intervals of depression.

And Terry, technically an advisor but truly an inspiration... When he agreed to work with me, I was crushed under the confusion of not knowing what to do after having invested years into the graduate school. He gave me the chance to work with him, to learn from him, and to engage in one of the most satisfying professions. If any of his relentless manner of thinking, his sheer drive to work, and his genuine interest to help people have rubbed off on me at all, then I know I'll be alright.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

I also would like to express my gratitude to my committee members Mike Schulte, Drew Kessler, Don Hillman, Dale Parson, and Samuel Gulden. These are five wonderful people who went out of their way to make things work out for me. Prof. Gulden was very instrumental in my admission to Lehigh University and my growth thereafter. Unfortunately he could not see my exit.

Finally, I feel obligated to mention the names of a few friends, teachers, colleagues, whose support meant beyond what they realize. I thank Adair Dingle and Madelene Spezialleti for believing in me early on, Steve Corbesero for trusting me with his machines, Jeanne Steinberg for helping me with the administrative side of the graduate program, Charles Kelemen and Lisa Meeden for allowing me to play in one of the most amazing playgrounds of education, Jeff Knerr for helping me with so many computing needs, Jim Wiseman and Matt Zukoski for being caring supportive friends, Ruth and Shelden Radin for being loving parents in-law, and my colleagues at Ithaca College for providing a welcoming environment to put this Ph.D. thing to use.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

For my parents and for Naomi...



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# Contents





Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# List of Figures





Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# Abstract

Hard real-time scheduling algorithms have their roots in the processor scheduling problem where a number of real-time tasks are coordinated to share the processor without missing any of their periodic deadlines. With the increase of continuous media applications, these techniques are now being considered to schedule the transmission of data streams across shared networks. From a theoretical standpoint, this is not surprising since the processor scheduling and the network scheduling problems are isomorphisms of each other. In both cases, there is a shared resource (the processor vs the network), there are concurrent tasks competing for the resource (the real time processes of the processor vs the real-time communication channels of the network), and there are deadlines associated with the completion of intermediate components (i.e. job completion times vs packet delivery times).

This dissertation is based on exploring the application of hard real-time scheduling algorithms to the network scheduling problem. We first show that the Distance Constrained Scheduling (DCS) algorithm can schedule a greater percentage of real-time task sets than Rate Monotonic Scheduling (RMS) algorithm even though DCS and RMS have the same worst case schedulability performance. We then develop a deterministic data-link later real-time protocol in order to compare their operational performance. We establish that DCS outperforms RMS in terms of the qualities of the produced schedules as well. Specifically, we show through simulations that DCS has better latency and jitter characteristics when scheduling traffic that we would associate with surveillance types of networks. The difference is especially significant when we incorporate system level variances in packet sizes, periodicity, and host synchronization during the formation of new schedules. The proposed data-link layer protocol also addresses many of the issues that need to be considered if real-time operation is to be achieved on non-real-time components.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# Chapter 1

# Introduction

The seminal paper "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment" published in 1973 [LL73] is widely viewed as the starting point of the theory of hard real-time scheduling. The Rate Monotonic and Earliest Deadline First scheduling algorithms designed by the authors have since then led to the creation of a growing set of mechanisms for coordinating real-time tasks on a single processor.

Since the 90s, interest in real-time scheduling techniques have risen even more due to the emergence of a similar and equally important problem: the coordination, delivery, and management of real-time network traffic. From a theoretical standpoint, this is not surprising since the processor scheduling and the network scheduling problems are isomorphisms of each other. In both cases, there is a shared resource (the processors vs the network), there are concurrent tasks competing for the resource (the real time processes of the processor vs the real-time communication channels of the network), and there are deadlines associated with the completion of intermediate components (i.e. job completion times vs packet delivery times)

Despite the similarities, however, numerous issues still wait to be resolved before the algorithms developed for processor scheduling can readily be applied to network scheduling. These issues, of course, are windows of opportunity for research. In fact, they form the basis of the work reported in this dissertation.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

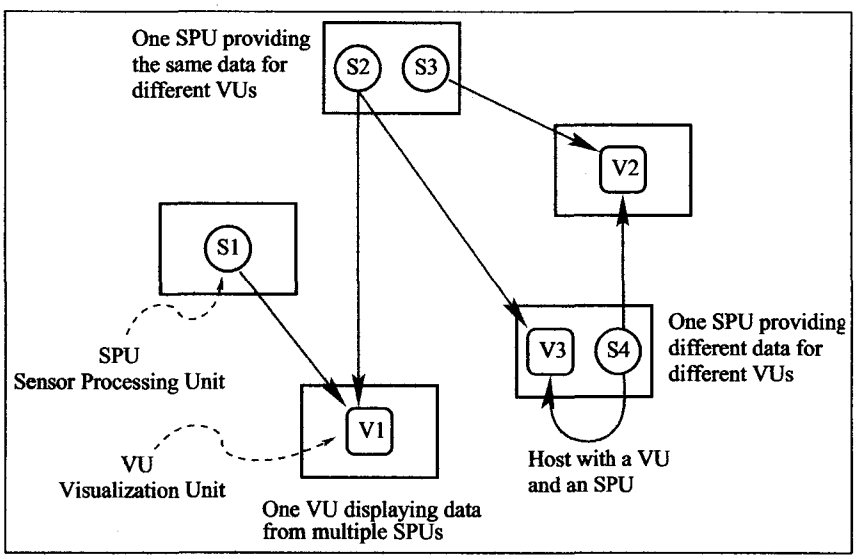



Figure 1.1: THE LOGICAL VIEW OF VSAM NETWORKING: *Each of the five enclosing rectangles represent hosts. A* ***Sensor Processing Unit*** *(SPU) is a host equiped with video capturing and streaming equipment. The* ***Visualization Units*** *(VU) are where the operators are situated. Each arrow represents a video stream whose frames need to be delivered under real-time constraints.*

## 1.1 Motivation

The driving reason for us to explore the application of real-time scheduling techniques in networking was to create a communication protocol for the Video Surveillance And Monitoring (VSAM) System developed by Terry Boult.

**Video surveillance** is the process of monitoring an area or a perimeter for significant events. Figure 1.1 shows the logical view of a video surveillance network where a few of the nodes are generating video streams while others are functioning as control nodes (potentially with human operators); a sample panoramic output of the VSAM system generated by one of the video nodes is shown in figure 1.2. Each video source generates a periodic stream of frames. As shown in figure 1.3, these frames are typically broken down into multiple packets due to the maximum transfer unit (MTU) sizes of the networking protocols involved.

One of the crucial factors in determining the effectiveness of a surveillance system is the timeliness (minimal latency/jitter) in which the video frames are delivered to the control



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 1.2: A PANORAMIC VIDEO FRAME FROM THE VSAM SYSTEM: *The rectangles are automatically superiposed on the frame sequence to help identify moving targets.*

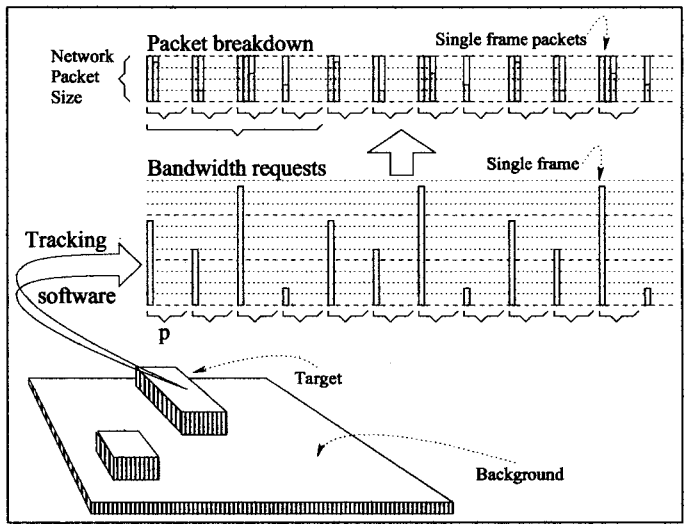



Figure 1.3: VSAM FRAMES TO PACKETS: *A video stream consists of periodically generated frames (or regions within these frames, as was shown with the rectangles in figure 1.2). Given typical maximum transfer unit sizes of common data link layer protocols, individual frames are typically broken into multiple packets and transmitted over the shared network to the associated control nodes.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

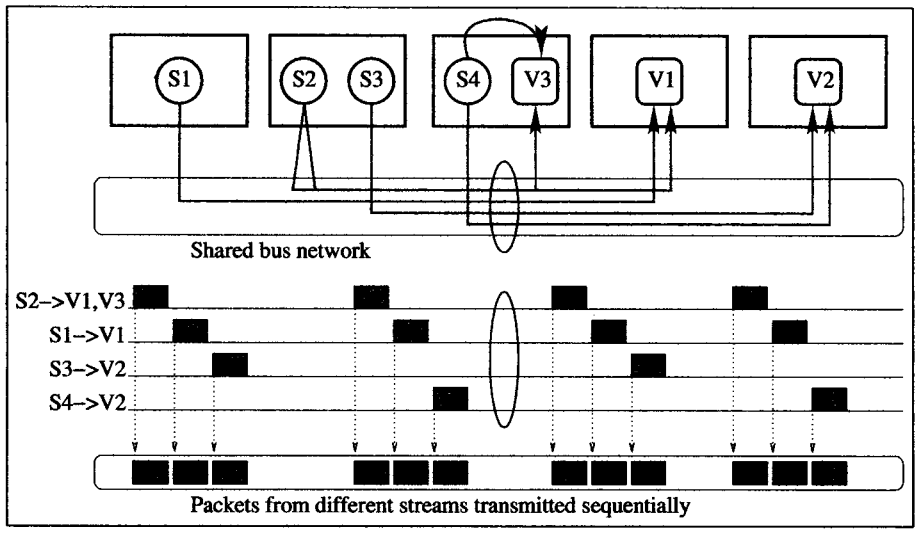



Figure 1.4: THE PHYSICAL VIEW OF VSAM NETWORKING: *The packets carrying the frames of the simultaneous video streams are transmitted over the same shared network. This means that the network is now a resource shared by an arbitrary number of periodic entities.*

nodes. This is not a trivial process since the point-to-point stream view of figure 1.1 is actually an illusion. As shown in figure 1.4, frames have to be transfered over a single shared network which creates a sequential bottleneck. Unless the networking infrastructure provides facilities for this bottleneck to be effectively managed, the performance of a surveillance network is hindered.

Data-link layer protocols such as 802.3 (Ethernet) and 802.11 (Wireless Ethernet) arbitrate network transmissions in a distributed and non-deterministic manner. This works well with non real-time applications but is not sufficient for video surveillance networks. First, conventional collision detection or avoidance mechanisms lead to under-utilization of the available bandwidth. In Ethernet, for example, sensing a collision, waiting a random amount time, and retransmitting creates a performance upper bound around 60% of the full transfer rate. In wireless Ethernet, the Request-To-Send and Clear-To-Send frames prevent collisions but consume network time (and thus bandwidth) of their own. Second, due to their randomized arbitration mechanisms, these protocols provide no upper bound for delays which get especially long as the network load is increased beyond trivial levels.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Third, their lack of admission control creates the possibility of the total networking demand to exceed 100% of capacity and to cause a performance collapse.

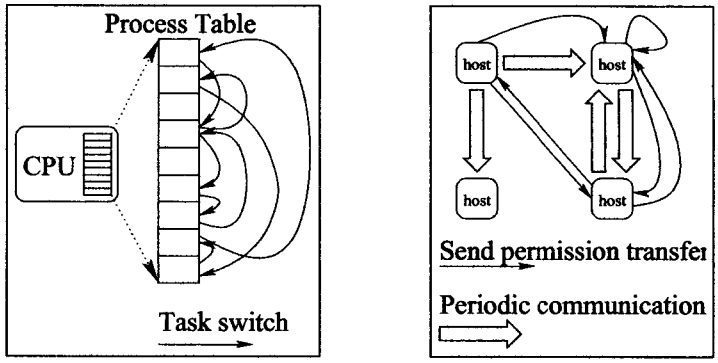
One solution to these problems is to devise a centralized data-link layer protocol to coordinate the transmissions of packets subject to delivery time constraints. In general, centralized solutions have limited value since they don't scale well. For surveillance networks, on the other hand, they seem to be suitable since the collective bandwidth intensity of video streams create an earlier upper bound on scalability. That is, individual network segments of a surveillance network are more likely to reach capacity due to the bandwidth limitations of the underlying network than they are due to the performance of the centralized scheduler.

A centralized scheduler provides a convenient point at which hard real-time scheduling algorithms can be utilized. Historically, these algorithms have been used for scheduling processors hosting concurrent real-time tasks. That is, they have been used to determine the order in which a single processor should be assigned among a set of tasks such that none of the tasks miss their periodically recurring deadlines. Our work is based on using the same algorithms to determine the order in which the shared network should be assigned to a set of periodic data streams; figures 1.5(a) and 1.5(b) graphically render the parallelism between these two applications.

In terms of our ongoing video surveillance network example, the scheme outlined above would mean that one of the nodes would have to function as the scheduler for the system (as discussed in later chapters, our protocol allows the scheduling responsibility to float among the participating nodes in order to prevent long term overloading of a single point). The scheduler would attempt to generate a new schedule each time a new video stream is created; failure to create a schedule (i.e. not being able to guarantee the timely delivery of the frames of the existing video streams) would cause the rejection of the new stream. If video streams are transmitted in compressed form and thus exhibit variable bit rate characteristics, then they would be split into constant bit rate substreams (as shown in figure 1.6) and scheduled as such.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.




(a) (b)

Figure 1.5: REAL TIME SCHEDULING: *Real-time scheduling algorithms are typically designed to coordinate the concurrent yet independent operation of tasks on a single processor. As shown in part 1.5(a), the chosen algorithm imposes a particular task switch order that guarantees the timely completion of each job of each task; this order is indicated by the thin arrows on the right side of the "zoomed up" representation of the CPU process table. The same idea is applicable in networks as well. Scheduling algorithms could be used to impose particular transmission orders among a group of hosts that periodically send messages. We graphically illustrate this idea in part 1.5(b) where the thick/gray arrows represent the transmission of messages and the thin lines represent the order in which these transmissions are made.*

We should also note that the benefits of a deterministic real-time data-link layer protocol is not limited to video surveillance networks alone:

- MONITORING SYSTEMS: Monitoring is the process of collecting runtime information from distributed applications so that developers can get a global view of the processes concurrently executing on different hosts ([SG94], [WSG98]). The usefulness of a monitoring system hinges on a number of factors. First, the global picture must be established from information gathered in a very timely manner; otherwise, the system can easily (and undetectably) create inconsistent views. Second, the operation of the monitoring system should influence the operation of the application by the smallest extent possible; otherwise, the global picture may not represent what actually happens when the application is running. We believe that the use of a real-time network protocol would help the design of monitoring system in both regards. If



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

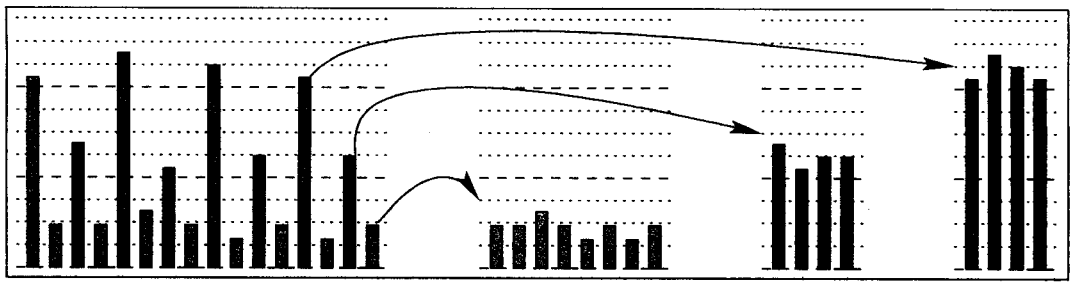


Figure 1.6: MPEG STREAM SUBSTREAMS: *The MPEG compression standard [Gal91] creates video streams that lead to variable bit rates, complicating the application of real-time scheduling algorithms. As shown in [KSH95], however, an MPEG stream can be split into three substreams (one for I, one for B, and one for P frames) each of which can be managed with a constant bit rate bandwidth reservation.*

the instrumentation data is collected via the periodic real-time streams, then their timely delivery would be guaranteed. In addition, the bandwidth allocations would isolate the traffic generated by the monitoring system from the traffic generated by the application, thereby reducing the networking portion of the "observation effect".

- ROUTING PROTOCOLS: Routing is arguably the most fundamental function needed by any non-trivial network. Though always a non-trivial problem, the recent incorporation of mobility has further complicated the creation of effective solutions. This is especially evident in mobile ad-hoc networks where quick expiration of topological knowledge requires a significant portion of the bandwidth to be consumed by routing updates. Again, we expect the availability of a real-time network protocol to have significant utility in this regard. The temporal guarantees would allow new routing protocols to place network-diameter based upper bounds on the propagation times of updates and the bandwidth allocations would prevent user applications from overloading the network and delaying these updates.

## 1.2 Project Goals

We focus on the single segment version of the network scheduling problem; that is, our protocol is designed for the scheduling of single collision domain. Consistent with the bulk



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of the scheduling literature, we assume that the periodic real-time streams carried through the network are independent of each other; that is, the period and packet sizes of one stream do not influence any other in any way. These packet streams start and stop at arbitrary points throughout the operation of the network.

### 1.2.1 Algorithmic Considerations

Historically, a significant portion of the hard real-time scheduling algorithms have been developed and studied for the purpose of never missing a deadline. Yet, the types of network applications that could make use of a real-time network protocol can certainly afford to miss the deadlines of a few of the countless packets they generate. Therefore, one of our goals is to reevaluate the existing algorithms to better understand their *average* case behavior rather than focus exclusively on their worst case limits.

In the simplest possible terms, average case analysis depends on the creation of a probabilistic framework in which the possible inputs to a problem are associated with likelihoods of occurrence. An expected value expression is then defined (in terms of the required work and the associated probability of each input) to quantify the amount of time and space needed to solve a "typical" problem. For example, when we study the performance of sorting algorithms, we first make various assumptions about the amount of randomness we find in the input. Then we create mathematical expressions that predict the number of numbers that need to be swapped for a total ascending or descending arrangement of the given values. This leads to final complexity measures such as $N \lg N$ that specify, on average, how much work is required to deal with $N$ numbers.

For many problems, however, the input is not as one dimensional as a sequence of numbers and the interaction of the different qualities of the input complicate the process of creating an analytical model. This appears to be the case with real-time scheduling. As will be discussed in chapter 3, there are too many crucial parameters that determine the performance of a scheduling system, making even the definition of "average" hard to pin down.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A second algorithmic issue is the characterization of the behavior of scheduling algorithms under uncertainty. In almost every scheduling algorithm surveyed for this project, the task set as well as the shared resource are defined in terms of constant parameters. Yet, nothing really remains constant in real networks for extended periods of time: the times at which the tasks request the resource change, the sizes of the requests vary around a mean, etc. This, of course, creates a gap between the pure theoretical systems on which algorithms are proposed and the "not-so-pure" real systems where these algorithms are needed.

As in many other complex problems that do not conveniently fit into an analytical model, our investigation of the above issues involves simulations. In fact, the bulk of information produced in this project come from simulations where we try to isolate and characterize numerous parameters of real-time scheduling. The ability to create, configure, and run a network protocol in a virtual network allow us to (i) examine the quality of the schedules produced by various algorithms, (ii) explore the consequences of probabilistic variations of system parameters, (iii) investigate the benefits of algorithmic adjustments in dealing with these variations. In the end, we show that even though two algorithms which we have used to drive the protocol scheduler have the same worst case characteristics, one seems to be far more applicable for network scheduling than the other.

### 1.2.2 Design Considerations

The scheduling of a network is different than the scheduling of a processor in one significant way: the source processes of the periodic streams execute on separate (and naturally independent) hosts, creating a number of communication problems that are altogether avoided in processor scheduling:

- In processor scheduling, the availability of a central clock is crucial in knowing where the system is with respect to knowing "who goes next". In network scheduling, time is available only in a distributed and unsynchronized way.

- In processor scheduling, the shared resource itself (i.e. the processor) does the pre-



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

emption of low priority tasks. In network scheduling, this is not possible since the network is a passive device. Therefore, the hosts of the segment must be explicitly notified of the scheduler's decisions.

- In processor scheduling, the overhead of task switching is small enough to be abstracted away. In network scheduling, task switching is a much more expensive operation since, as we will see in chapter 4, it requires the transmission of a token packet. But the real problem is the dynamic nature of this overhead. First, we assume that we are dealing with a heterogeneous network of hosts and thus the addition of a constant value to each task switch is not a suitable solution. Second, these hosts are expected to be running an arbitrary set of user applications which effect their responsiveness. Therefore, even a different yet constant task switch overhead per host is not flexible enough to provide an accurate task switch model.

A more fundamental consequence of the distributed nature of network scheduling surfaces when we consider the operation of the scheduler itself. As discussed in chapter 2, both static and dynamic scheduling algorithms assume that the scheduler has preemptive control over the use of the shared resource. In the case of static scheduling algorithms (Rate Monotonic Scheduling, Distance Constrained Scheduling), the priority assignments that determine the actions of the preemptive scheduler get set at the beginning of the schedule while those of dynamic scheduling algorithms (Earliest Deadline First, Deadline Monotonic Scheduling) can vary for each job. But for both classes of algorithms, the scheduler needs to know the set of currently ready/waiting jobs to pick the one with the highest priority. If this information is not instantaneously and accurately available, then the performance of underlying algorithm drops even below the worst case level guaranteed by mathematical arguments/models.

It is conceivable that if the hosts and the periodic streams operate with utmost precision and do not deviate from the values of the constant parameters that define them, then a truly distributed dynamic scheduling algorithms can be found. However, in a real network,



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

this is too luxurious an assumption; non-real-time multi-tasking operating systems prevent the establishment of precise timing information even on individual hosts. For these reasons, static scheduling algorithms with their smaller dependency on instantaneous scheduling information appear to be a more realistic and practical choice for our goals.

The greatest factor in the timeliness of the state information collected by the network protocol is naturally the method with which the transmission medium is accessed. During the past two decades, there have been a great number of protocols ranging from IEEE standards to experimental systems to establish simple and efficient mechanisms for this purpose. As surveyed at the beginning of chapter 4, CSMA and the CSMA-derived protocols have explored distributed schemes which have led to the most prevalent data-link layer protocols. Unfortunately, their non-deterministic design (which is the basis of their distributed operation) makes them unsuitable for our purposes; if transmission time of a packet is based on a pseudo-random function that depends on the transmission of other hosts at the same time, then real-time guarantees at higher levels cannot be established. The deterministic protocols, on the other hand, have either required particular wiring (Token Ring: IEEE 802.5), or been too complex (Token Bus: IEEE 802.4) or too application specific (Rether: [VcC95], [cCV97b], [cCV97a], [PC98]). For these reasons, the operation of our real-time protocol makes use of a new token based scheme that arbitrates media access is a deterministic way. Though quite simple by design, it is unique and (to the best of our knowledge) novel in the sense that its operation is tightly coupled with the coordination of a high level scheduler. It is through the bandwidth allocations achieved by this scheme that the network protocol can establish temporal correctness for the periodic streams and a best-effort delivery mechanism for the aperiodic transmissions.

### 1.2.3 Mathematical/Theoretical Considerations

The most significant accomplishments in the history of real-time scheduling have been mathematical. Almost without exception, each of the well-known hard real-time scheduling algorithms has been accompanied by a utilization schedulability threshold (and its proof)



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

such that all task sets whose combined utilization is lower than this threshold is guaranteed to be schedulable by that algorithm; the survey in chapter 2 has the associated details. Recently, however, as real-time systems are applied in new systems that are more tolerant of occasional deadline failures, the focus on the establishment of scheduling has shifted more toward statistically determined behavior.

A significant portion of the effort involved in this project was indeed within this mathematical plane. Specifically, we considered probabilistic formulations of the problem to establish tighter statistical guarantees for the underlying scheduler. Unfortunately this also turned out to be the area in which we made the least progress. What we report here is limited to a general description of our attempts as well as the outline of a statistical schedulability limit for distance constrained scheduling.

## 1.3 Thesis Outline

Chapter 2 provides a survey of various real-time scheduling techniques, most of which had originally been designed to schedule processors. Since this area of research has been very active during the past three decades, there is much ground to cover and thus our focus is limited in two ways. First, we consider algorithms that have the potential to be used in network scheduling. Second, we consider static scheduling algorithms even though dynamic algorithms are known to have better utilization rates. As noted earlier, the primary reason for this is that the distributed nature of the network scheduling problem prevents the timely collection of state information from the source hosts during the operation of the real-time protocol. This in turn prevents dynamic scheduling algorithms to outperform static ones.

In chapter 3, we narrow our focus on two scheduling algorithms that have the potential to drive a real-time network protocol: **rate monotonic** and **distance constrained** scheduling. We start by enumerating several parameters that determine the schedulability characteristics of task sets and we perform a series of simulations. Our goal here is to isolate the significant factors so that later simulations can focus on the ones that make notable



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

differences.

In chapter 4, our focus shifts from algorithmic considerations to practical ones. We review the virtual environment we have created to study the performance of different scheduling schemes. We then discuss an algorithm independent network protocol that offers a pseudo real-time service based on average host responsiveness and bandwidth allocations. Although this network protocol is tested only within the virtual environment, every one of its design decisions are made to make it feasible on actual systems.

Chapter 5 is exclusively based on rate monotonic scheduling. After a discussion of the algorithm specific issues within the protocol, we present the results of a large array of simulations that illustrate the performance of this scheme, specifically in terms of its ability to handle variations in the resource sizes and ready times. Likewise, chapter 6 is based on the results of the distance constrained scheduler. Aside from the performance summary of this algorithm, we suggest a number of modifications to its basic operation that boosts its performance in dealing with varying system parameters.

Finally, chapter 7 provides a summary of our results as well as a general discussion of some of the mathematical and implementation issues we plan to tackle in the near future.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# Chapter 2

# Real-Time Scheduling

This chapter is a survey of periodic real-time scheduling techniques. Since the existing body of literature on this topic is vast, we limit our presentation to those techniques that can (at least potentially) be applied to the network scheduling problem. Specifically, we present systems where (i) tasks are periodic, (ii) tasks are associated with hard/firm deadlines, (iii) tasks are competing to share a single resource, (iv) there are no precedence constraints between the tasks, and (v) tasks do not share data. We also limit our focus on static scheduling methods. Though not as capable as dynamic scheduling, we consider static scheduling to be more appropriate for network scheduling; the details and justification of this presented are later in this chapter.

A brief introduction is followed by section 2.2 where we define the terms that are frequently encountered in the real-time scheduling literature. Since the starting point for any theory is an abstraction of the associated real life system, section 2.3 presents the task models on which all well-known scheduling algorithms are formulated. Sections 2.4 and 2.5 present the periodic and the distance constrained scheduling models. This is followed by a short section on dynamic scheduling algorithms (which is included only for the sake of completeness; the overall emphasis remains on static algorithms). We conclude this chapter after a list of some of the more recent efforts to generalize resource requests in section 2.4.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

## 2.1 Introduction

In non real-time systems, "correct output" means correctness in the mathematical sense alone; the time it takes to compute this output is usually considered as an asymptotic complexity issue of the underlying algorithm. In real-time systems, on the other hand, the time it takes to complete a computation becomes as important as its mathematical correctness: a result produced beyond the completion deadline of a problem has no more value that a mathematically incorrect one.

Historically, temporal correctness (i.e. the ability to compute a solution within its deadline) has been occasionally treated as an afterthought for systems that were meant to operate under real-time constraints. Specifically, the operation of systems based on non real-time designs have later been tuned so that intermediate and/or final results are computed within time-bounds. As explained by Sha and Goodenough in [SG90], such ad-hoc methods have characteristically produced systems where a slight change in the specification require the difficult re-adjustments of many timing parameters[1]. Ideally, the process of assuring temporal correctness should be based on well-understood methods that can be automatically imposed by the system itself, freeing the designer to concentrate on mathematical or functional correctness.

Today, real-time theory offers a suite of scheduling algorithms for temporal correctness. It has been applied to a diverse set of systems ranging from real-time operating system kernels to coordination schemes for tuning on signals from non-geostationary satellites [HMR+89]. In all these cases, as long as the system functions within specific resource request bounds and as long as its behavior is (at least partially) periodic, all deadlines are guaranteed to be met automatically.

[1]The authors draw an illustrative analogy between ad-hoc temporal correctness solutions and old fashioned memory overlay schemes. In both cases, designers spend too much time working with low-level details "to get the system right" and still not have much confidence in the final product.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

## 2.2 Basic Definitions

The field of real-time systems can be broken down into a number of areas (fault-tolerance, redundancy, etc) and a device independent study of any one of them requires a suitable form of abstraction. When the focus is on scheduling, real-time systems are almost always represented with task sets. A **task** corresponds to a process or a thread or any other form of computation that operates under time constraints. In general, there could be any number of processors serving the task set. However, in our area of interest, we make the assumption that the task set is operating on a single processor, the one key resource for which the members of the task set compete.

**Periodic real-time scheduling** is a specialized form of real-time scheduling where tasks exhibit repetitive behavior. At the start of each period, each task requests the execution of a block of code within a specific interval of time. Once the request is complete, the task suspends itself until the beginning of the next period. This repetitiveness is crucial in making predictions for future resource requests.

The **scheduler** component of a real-time system follows the algorithmic logic of a scheduling technique to determine the order and the duration of which task gets to use the processor. In many cases, the scheduler's work reduces to making priority assignments: whenever multiple tasks simultaneously request the shared resource, the scheduler simply picks the one with the highest priority.

The consequences of temporal failures provides another common way of classifying real-time systems. With **hard real-time systems**, all the deadlines have to be met since missing even just one could be "fatal". If, however, occasional deadline failures can be tolerated, most hard real-time systems find no value in late computations and thus abort jobs that are certain to exceed their resource request window. With **soft real-time systems**, missing a deadline is "undesirable". In addition, unlike hard real-times, tasks are assumed to have value left in them even beyond their expected completion time. Consequently, soft real-time schedulers attempt to run their tasks to completion even when some will unavoidably be



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

late.

Some authors (as in [Bur91]) make use of a **utility** measure to quantify the difference between hard real-time and soft real-time systems. The utility of a task represents its usefulness where positive values indicate progress/benefit, 0 indicates indifference, and negative values indicate damage. In soft real-time systems, the utility of a task gradually drops beyond a missed deadline, eventually reaching down to zero. In hard real-time systems, the utility becomes highly negative immediately after a deadline.

Quantitative comparisons of the capabilities of real-time scheduling algorithms hinge on the task models on which these algorithms are defined. A great number of proofs in this context start by defining a **scheduling domain** which is an uncountably infinite set of task sets that can be described by a particular task model. For example, if tasks are modeled with only a pair of numbers {request-length, period-length}, then the associated scheduling domain contains all task sets of a single task, all task sets of two tasks, all task sets of three tasks, etc, where each task is a reasonable pair of numbers to define a task[2]. These proofs then proceed to define a subset of the scheduling domain that can be successfully scheduled with a particular algorithm. Or, they prove whether a scheduling algorithm can schedule the portion of the scheduling domain covered by a competing or baseline algorithm.

There are cases where one algorithm is established as the most capable one for its associated task model. In the real-time scheduling literature, these algorithms are said to be **optimal**[3]. By definition, if an optimal algorithm fails to schedule a task set, then no other algorithm based on the same task model can possibly schedule that same task set. For this definition of optimality, it is irrelevant to consider *by what margin* the system is able to meet its deadlines. The critical consideration is *whether* the system is able to meet them. It may seem that optimality in this context is being defined in a rather unusual way. However, the definition is indeed consistent with the framing of real systems: the

[2] "Reasonable" in this case might mean that request-length is greater than 0 and periodic-length is greater request-length.

[3] It is important to note the contrast between this definition of optimality vs its usual meaning. The former is mostly a matter of subset coverage while the letter simply implies the "goodness" of a particular solution (as in "the optimal path between two cities").



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

fundamental distinguishing quality for real-time systems is that their success is based not on speed (though speed usually does not hurt) but their ability to meet deadline constraints.

If a schedule can be created for a given task set, then the task set is considered **feasible** (or **schedulable**) and the schedule is considered **valid**. Although these definitions seem fairly straight forward, the terms are occasionally used in ways that seem confusing. For example, feasibility can be used in a relative way as in "a task set that is *infeasible* for static schedulers might be *feasible* for dynamic schedulers" (as was mentioned earlier, dynamic schedulers are more capable than static schedulers at the cost of increased operational overhead and complexity).

## 2.3 Task and Scheduler Models

The task model is a fundamental component of the abstraction of a real-time system. The features captured in this model determine the applicability, sophistication, and complexity of the scheduling analysis.

As mentioned earlier, a real-time system hosts a set of $n$ processes, threads, communication streams, etc., that are generically referred to as **tasks**. Task $\tau_i$ follows a repetitive pattern of requesting the use of a shared resource such as a processor or a communication channel[4]. Consequently, the timeline of a task consists of alternating intervals of resource request and idle time. Each resource request interval of $\tau_i$ is referred to as a **job** of $\tau_i$. Jobs[5] are identified by subscripting the name of the task they belong to: the $j$th job of task $\tau_i$ is $\tau_{i,j}$. Function $\mathrm{R}(\tau_{i,j})$ is the **ready time** of job $\tau_{i,j}$ and specifies the earliest moment $\tau_{i,j}$ can start using the shared resource. However, the actual starting time of $\tau_{i,j}$ may be later than $\mathrm{R}(\tau_{i,j})$ if other jobs are currently using the shared resource and $\tau_{i,j}$ does not have the required precedence to preempt them. Function $\mathrm{F}(\tau_{i,j})$ is the **finish time** of job $\tau_{i,j}$ and specifies the moment $\tau_{i,j}$ reaches completion. Unless multiple jobs of a single task are

[4]In this document, indexes and subscripts count from 0. That is, $n$ elements are enumerated with the numbers $0 \,..\, (n-1)$ instead of $1 \,..\, n$.

[5]The term "job" is an artifact of the early Operations Research roots of real-time scheduling. In some recent papers, more modern terms such as "instance" or "computation" have been used but we prefer to stick with the original.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

allowed to coexist, a sequential order is imposed on jobs: $\mathrm{F}(\tau_{i,j}) \leq \mathrm{R}(\tau_{i,j+1})$.

In order to arbitrate resource access, tasks and their jobs are associated with priorities to determine who gets to use the resource when multiple tasks simultaneously make a request for it. We use the notation of $\rho(\tau_i)$ and $\rho(\tau_{i,j})$ to represent the priorities of task $\tau_i$ and job $\tau_{i,j}$, respectively. With **static priority scheduling** (or just "static scheduling"), every job of task $\tau_i$ is assigned the same priority; that is, if $\rho(\tau_i) = x$, then $\rho(\tau_{i,j}) = x$ for all $j \geq 0$. With **dynamic priority scheduling** (or just "dynamic scheduling"), different jobs of a task may be assigned different priorities based on the instantaneous conditions of the task set and the underlying system; that is, $\rho(\tau_{i,j})$ and $\rho(\tau_{i,j+1})$ do not have to be the same [LL73], [HL92]. Due to their greater flexibility, dynamic scheduling algorithms achieve higher utilizations, but static scheduling has its advantages as well. With fixed priorities, scheduling is easier to analyze and thus its theory is more complete. Static scheduling is robust under conditions of transient overload. It has been extended in various directions such as accommodating aperiodic tasks, preventing priority inversion, incorporating imprecise computation, etc. It is also simpler to implement and causes less overhead in general.

**Offline** scheduling systems (AKA **fixed** systems) assign priorities to task slots of the processor *before* the resource requests and completion deadlines of the task set are known. **Online** scheduling systems assign task priorities *after* the scheduler sees the task set [Bur91]. Occasionally, the phrases "static schedulers" and "offline scheduler" are incorrectly stated to mean the same thing. As noted in, [SSDB95] static schedulers *do* make scheduling decisions at run time but this run-time information is not utilized for reassigning priorities after the initial assignments have been made. Therefore, both static and dynamic scheduling systems are considered to be online.

Much like the case in more general scheduling systems, real-time schedulers operate either **preemptively** (where a task can be taken off the resource at any moment by the scheduler) or **non-preemptively** (where the scheduler always waits for the resource to be returned by the currently executing task before a different task can be assigned to the resource). In general, the ability to preempt leads to better performance since the sched-



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

uler never has to wait to reassign the resource to improve resource utilization. However, exceptions are noted in [SS94]. Furthermore, preemptive scheduling involves more computational overhead and is relatively more complex to implement. In fact, if synchronization among the tasks is required (which, for the record, is an issue we have assumed out of our considerations), preemptive operation may even turn out to be impossible.

For most models, it is assumed that the underlying system can provide a unique priority level for each task. If however, the number of priority levels is lower than the number of tasks, then maximum achievable utilization of the shared resource may be reduced. In addition, if the tasks are communicating with synchronized access through shared memory, then a priority level shortage could to lead to cases of **priority inversion** where a low priority task prevents the progress of a high priority task by locking a write-synchronized resource [LS86], [GS88], [SRL90], [BW91], [KSS95].

The regularity with which jobs are created is another factor of classification for periodic real-time tasks. With **sporadic tasks** ([Mok83] and [HLH96a]), consecutive jobs of a task are subject to a **minimum separation** constraint so that $\mathrm{R}(\tau_{i,j+1}) - \mathrm{R}(\tau_{i,j})$ is greater than or equal to a fixed value (the minimum separation constraint is a necessity since it puts a bound on maximum resource demand). It is also possible to define a **maximum separation** constraint so that minimum resource requirements can be determined.

**Periodic tasks** are a special case of the sporadic tasks where the minimum separation constraint is equal to the maximum separation constraint. A substantial body of literature exists for periodic tasks for three reasons: (i) the operation of many actual real-time systems can be modeled with periodic tasks, (ii) the regularity of the creation of jobs simplifies scheduling analysis, and (iii) since periodic tasks are the limiting case for sporadic tasks, most results developed for the periodic model can be applied to sporadic tasks for worst case analysis. **Distance constrained tasks** is a more recent model than periodic tasks. Its main advantage over the periodic model is that it reduces the scheduling **jitter** which is defined as the variation in the sequence of $\mathrm{F}(\tau_{i,j}) - \mathrm{R}(\tau_{i,j})$ values[6].

[6]The Distance constrained tasks are more "periodic" than periodic tasks are. Unfortunately, the term



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Every scheduling algorithm associates a deadline $d_{i,j}$ with a job $\tau_{i,j}$ though they differ by the stringency that applies to $d_{i,j}$. In the simplest case, a deadline is aligned with the ready time of the consecutive job; that is, $\mathrm{R}(\tau_{i,j}) + d_{i,j} = \mathrm{R}(\tau_{i,j+1})$. A more general approach is to allow $\mathrm{R}(\tau_{i,j}) + d_{i,j} \leq \mathrm{R}(\tau_{i,j+1})$ which creates a potentially smaller time window for jobs to reach completion. When the possibility of $\mathrm{R}(\tau_{i,j}) + d_{i,j} \geq \mathrm{R}(\tau_{i,j+1})$ exists, multiple jobs of a task may exist simultaneously which opens up queuing issues [LL73]. Some models use job overlapping for leveling out the peaks and valleys of aggregate resource request. Overall, however, the scheduling theory is more complete for the $\mathrm{R}(\tau_{i,j}) + d_{i,j} \leq \mathrm{R}(\tau_{i,j+1})$ case.

## 2.4 Scheduling Periodic Task Models

As shown in figure 2.1, a **periodic task** is defined as $\tau = (p, c, s, d)$ where $p$ is the **period**, $c$ is the **resource request**[7], $s$ is the **start time**, and $d$ is the **deadline**. When we deal with task sets, we subscript the parameters and thus have $\tau_i = (p_i, c_i, s_i, d_i)$ for the $i$th task. The time window for job $\tau_{i,j}$ is from $\mathrm{R}(\tau_{i,j}) = s_i + jp_i$ to $\mathrm{R}(\tau_{i,j}) = s_i + jp_i + d_i$ for $0 \leq i < n$, $j \geq 0$. In order to prevent queuing issues in job completions, most models require $d_i \leq p_i$.

Three simpler versions of the periodic model are frequently used in the literature. When the $s_i$ parameter is eliminated, all tasks are assumed to be submitted to the system at time $t = 0$. When the $d_i$ parameter is eliminated, the deadline of a particular job is the ready time of the successor job. Finally, when both $s_i$ and $d_i$ are dropped, we have the simplest (and probably the most frequently used) version of the periodic task model as shown in figure 2.2. In [LW82], the two models with an explicit start parameter $s_i$ are referred to as **asynchronous** models while the two without a start parameter are referred to as the **synchronous** models; as we will see later in this section, synchronicity is a significant factor in simplifying schedulability analysis. The table in figure 2.3 summarizes the four variations of the periodic task model.

"periodic tasks" had already been defined and used consistently since the 1970s and the introduction of the distance constrained model must have required a different name to avoid confusion.

[7]Variable name $c$ is from the existing scheduling literature which refers to resource request as "computation". In addition, it is expressed in units of time to be compatible with the other three parameters.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

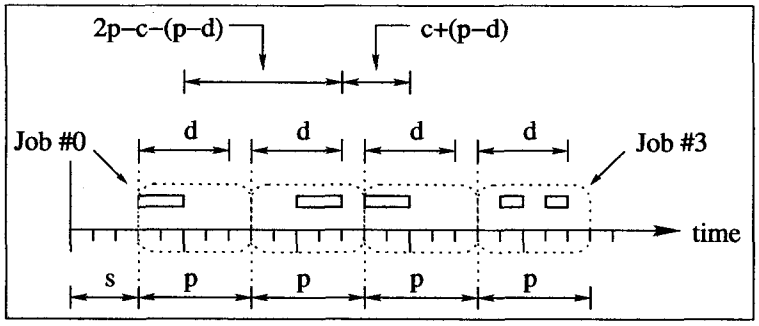



Figure 2.1: PERIODIC TASK MODEL: *In its most general form, task $\tau$ is represented with a four-tuple that states the period p, resource request c, start time s, and deadline d using the notation $\tau_i = (p, c, s, d)$. In this diagram, we have $\tau_i = (5, 2, 3, 4)$ where the darkly shaded rectangles represent the resource request while the lightly shaded (larger, rounded) rectangles show the job boundaries. Job #3 (counting from 0) is fragmented due the preemption of higher priority task (though we do not see that task here to keep things simple).*

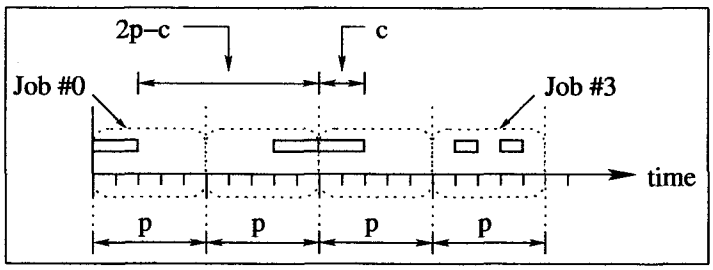



Figure 2.2: SIMPLIFIED PERIODIC TASK MODEL: *Without the start and the explicit deadline parameters, a periodic model task reduces to a pair of numbers $\tau_i = (p_i, c_i)$. In this example, we have $\tau_i = (5, 2)$.*

Two characteristics of the periodic task model need to be stressed. First, once the $p_i$, $c_i$, $s_i$, and $d_i$ parameters of a task are defined, the time windows for *all* jobs of that task are set since the task timeline is partitioned into fixed size intervals. This is different than other models where the time window for job $\tau_{i,j+1}$ is known only after $\tau_{i,j}$ has completed. Second, representing the resource request with a single $c_i$ value may or may not be realistic enough. A constant $c_i$ is an acceptable approximation for the tasks of many control and monitoring type real-time systems which have uniform resource request sizes. This also simplifies the scheduling analysis and leads to efficient admission tests for new tasks. However, resource requests in the general case do not have a uniform size. For example, a compressed



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

| Name | Type | Tuple | Job Window | Restrictions |
|---|---|---|---|---|
| PC | Synch | $\tau_i = (p_i, c_i)$ | $jp_i \,..\, (j+1)p_i$ | $c_i \leq p_i$ |
| PCS | Asynch | $\tau_i = (p_i, c_i, s_i)$ | $s_i + jp_i \,..\, s_i + (j+1)p_i$ | $c_i \leq p_i$ |
| PCD | Synch | $\tau_i = (p_i, c_i, d_i)$ | $jp_i \,..\, jp_i + d_i$ | $c_i \leq d_i \leq p_i$ |
| PCSD | Asynch | $\tau_i = (p_i, c_i, s_i, d_i)$ | $s_i + jp_i \,..\, s_i + jp_i + d_i$ | $c_i \leq d_i \leq p_i$ |

Figure 2.3: COMPARISON OF THE FOUR TASK MODELS: *The letters 'P', 'C', 'S', and 'D' in the abbreviated names stand for "period", "resource request", "start", and "deadline", respectively.*

video stream displays a high degree of variation in its bandwidth requirement. Although a constant $c_i$ could be interpreted as a worst case condition, the resulting scheduling analysis would be pessimistic since the average demand is usually much lower. Admission tests based on worst case resource usage will reject many new tasks that would have been schedulable if a more accurate usage pattern had been available. As outlined in section 2.7, a significant portion of the recent research in periodic real-time scheduling is to model variable resource requirements while retaining the level of predictability with constant $c_i$'s.

Before we describe the primary periodic task scheduling algorithms, we define a few functions (often used in the literature) to simplify subsequent explanations. The $p$ subscript of task set names (as in $\mathrm{T_p}$) is to remind the reader that we are dealing with periodic task sets rather than, for example, distance constrained or pinwheel task sets:

- The **utilization** of a periodic task is the ratio of its resource request to its period: $\mathrm{U}(\tau_i) = \frac{c_i}{p_i}$. The utilization of a periodic task set $\mathrm{T_p}$ is the sum of the utilizations of the member tasks: $\mathrm{U}(\mathrm{T_p}) = \sum \mathrm{U}(\tau_i)$ for $\tau_i \in \mathrm{T_p}$.

- A **scheduling instant** is a point in time when a job is submitted to the system. For a PCSD task $\tau_i$, the scheduling points are at $t = s_i + jp_i$, $j \geq 0$. For a set of tasks, the scheduling instants are the union of the scheduling instants of the member tasks. For example, for $\tau_0 = (p_0 = 3, c_0, s_0 = 2, d_0)$ and $\tau_1 = (p_1 = 5, c_1, s_1 = 2, d_1)$, the scheduling instants are at $\{2, 5, 7, 8, 11, 12, 14, 17, 20, 22, ...\}$ which is the union of $\{2, 5, 8, 11, 14, 17, 20, ...\}$ (from $\tau_0$) and $\{2, 7, 12, 17, 22, ...\}$ (from $\tau_1$).

- The **load function** $\mathrm{L}(\tau_i, t)$ specifies how much work has been submitted to the



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

system by task $\tau_i$ during time interval $[0 .. t]$. $\mathrm{L}(\tau_i, t)$ is a monotonically increasing function whose value goes up by $c_i$ at every scheduling instant of $\tau_i$: $\mathrm{L}(\tau_i, t) = c_i\lceil\frac{t}{p_i}\rceil$. For a task set, $\mathrm{L}(t) = \sum \mathrm{L}(\tau_i, t)$.

- The **progress function** $\mathrm{P}(\tau_i, t)$ specifies how much resource $\tau_i$ has used in the interval $[0 .. t]$, assuming that no deadlines have been missed. Regardless of the scheduling algorithm used, $\mathrm{P}(\tau_i, t)$ is always contained in the range $c_i\lfloor\frac{t}{p_i}\rfloor .. c_i\lceil\frac{t}{p_i}\rceil$. For a task set, $\mathrm{P}(t) = \sum \mathrm{P}(\tau_i, t)$.

- The **work function** $\mathrm{W}(\tau_i, t)$ specifies how much work is yet to be done by task $\tau_i$ at time $t$. Assuming $\mathrm{P}(\tau_i, t)$ is defined at time $t$, $\mathrm{W}(\tau_i, t) = \mathrm{L}(\tau_i, t) - \mathrm{P}(\tau_i, t)$. For a task set, $\mathrm{W}(t) = \mathrm{L}(t) - \mathrm{P}(t)$.

We make the following observations about the work function $\mathrm{W}(t)$:

- $\mathrm{W}(t)$ is periodic. This is because periodic task systems are by definition "closed" (i.e. the set of the predetermined periodic tasks are the *only* elements of resource request) and the constituent tasks are themselves periodic (which means their "work sum" will also be periodic). Each cycle/period of $\mathrm{W}(t)$ is generally referred to as a **major cycle** of the schedule. The length of the major cycle is at most the **least common multiple** (LCM) of all the periods of the tasks in the set.

- $\mathrm{W}(t)$ is 0 at those instances when the system has caught up with all the submitted work and $\sum c_i$ whenever all tasks simultaneously submit instances. At all other times, the value of $\mathrm{W}(t)$ is contained between these two extremes.

Given the periodicity of $\mathrm{W}(t)$, the schedulability and scheduling problems for *synchronous* periodic tasks can be solved by attempting to create a schedule for the first major cycle of the task set using an optimal algorithm. If all the deadlines are met, then the task set is schedulable and the system can repeat this schedule indefinitely or until a request for a new schedule is received; otherwise, the task set is not schedulable (this morbid conclusion can readily be made since we used an *optimal* algorithm in our attempt to create



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the schedule). The minimum necessary and sufficient interval to establish schedulability for *asynchronous* task sets, on the other hand, is shown to be $[0 \,..\, \max\{p_i\} + 2\,\text{lcm}\{p_i\})$ in [LW82], which is roughly twice as long as the synchronous case.

For the PC class of tasks (i.e. those that are defined only by a period and a resource request), [LL73] and [LW82] prove that the validity of the schedule in the interval $[0 \,..\, \max\{p_i\})$ is a necessary and sufficient condition for the validity of schedule in $[0 \,..\, \text{lcm}\{p_i\})$. This is a consequence of having all the tasks start at time $t = 0$ thereby creating the greatest jump in the load function right at the beginning of the schedule. If the system can "survive" this jump by meeting the first deadline of each of the tasks, then it can meet all the deadlines of that task set during any interval.

A feasibility test based on the construction of a schedule over the interval $[0 \,..\, \max\{p_i\})$ is known as the **critical region** test and is used (i) in [LL73] for PC tasks, (ii) in [LW82] for PCD tasks, and (iii) in [MC96b] for multi-frame tasks (which is covered in section 2.7). The critical region test for synchronous tasks provides a substantial reduction in the complexity of the schedulability analysis. In the general case, the length of a major cycle (i.e. the LCM of the periods) can be as high as the product of all the task periods. However, the length of the partial schedule required by the critical region test is the longest period in the set.

In [LL73], **Rate Monotonic Scheduling** (RMS) is proposed as an optimal static scheduling algorithm for PC tasks. The RMS priority assignment is based on a single criteria: if $p_i < p_j$, then $\rho(p_i) > \rho(p_j)$; that is, the shorter the period of a task, the higher the priority it gets[8]. Given a set of $n$ PC tasks, RMS can *always* create a schedule as long as the total utilization is at most $\text{U}(n) = n(2^{\frac{1}{n}} - 1)$; this upper limit is a monotonically decreasing value from 0.83 (when $n = 2$) to $\ln(2) \approxeq 0.693$ (as $n$ tends toward $\infty$). If the total utilization $\text{U}(n)$ is in the range $(n(2^{\frac{1}{n}} - 1) \,..\, 1]$, then the critical region test is used as a schedulability test. In [LSD89], an efficient implementation of the critical region test is given.

[8]The **rate** of a task is the reciprocal of its period. **Monotonicity** implies either "nonincreasing" or "nondecreasing" (i.e. a monotonic function has no local minima or maxima). Based on these, the RMS priority assignment function *monotonically* increases with the *rate* of the task it is applied to [War91].



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

RMS is one of the most successful results of real-time scheduling theory. Many task models other than the synchronous PC model use it to assign task priorities (distance constrained, multi-frame, statistical RMS, etc). It is also frequently used as the underlying scheduling algorithm when other aspects of real-time scheduling are investigated (e.g. priority inversion ([GS88], [SRL90], [BW91]), mode changes ([SRLR89], [TBW92], etc).

When the task model is PCD rather than PC, RMS is no longer optimal; in fact, the use of deadlines shorter than task periods mislead RMS into assigning priorities which sometime result in invalid schedules for schedulable task sets. However, **Deadline Monotonic Scheduling** (DMS), which is a slight modification of RMS, is optimal for the PCD model [LW82]. The DMS priority assignment is based on deadlines instead of periods: if $d_i < d_j$, then $\rho(p_i) \geq \rho(p_j)$; as with RMS, the critical region analysis is used as a valid schedulability test. In [HLH96a], it is stated that the DMS can be used to schedule sporadic tasks. This is a consequence of the fact that the PCD model is the "densest" form of the sporadic task model where task instances are arriving at the fastest possible rate (i.e. instance separation is always equal to the minimum separation constraint).

One of the shortcomings of RMS and DMS algorithms is the determination of task priorities from period/deadline values alone. During periods of temporary overload, deadline failures start with the tasks with the longest periods/deadlines regardless of their actual utility to the system. One solution is the technique of **period transformation**[9] where the $p_i$, $c_i$, and $d_i$ parameters of a task are reduced by an amount proportional to the desired increase in importance ([SLR87], [SLR87], [SRLR89]). Another method is to establish a total ordering within the task set based on importance independent of the scheduling priorities. Depending on the severity of the overload, the system is then expected to apply the RMS/DMS algorithms to the first $x$ tasks by this order.

Figure 2.4 summarizes the known optimal algorithms for scheduling the four types of periodic tasks under the single resource formulation of the repetitive scheduling problem.

[9]The **specialization techniques** of pinwheel scheduling (explained later in this section) is a type of period transformation. Therefore, period transformation is a tool that can be used with distance constrained tasks as well as periodic tasks.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

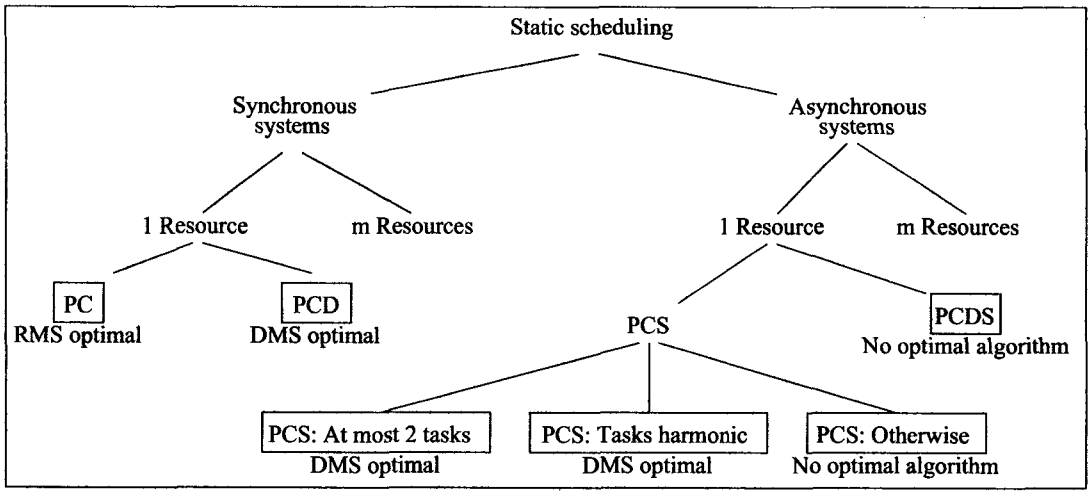



Figure 2.4: ALGORITHMIC BREAKDOWN: *For the single resource sharing problem, RMS is optional for the PC model while DMS is optimal for the PCD model. DMS is also optional for two special formulations of the PDC model. No optimal priority assignment algorithm is known for the PCDS model. The multiple resource case (i.e. the "m resources" nodes above) is not covered in this survey. Also, for the record, we reiterate here that the notions of "schedulability" and "feasibility" depend on the scheduling method used. For example, some task sets that were considered infeasible with static scheduling* are *feasible using dynamic scheduling. Consequently, the above breakdown of schedulability results are not applicable when dynamic scheduling is considered.*

As stated before, these results should be evaluated within the scope of static priority assignments since they do not automatically apply to dynamic priority assignments.

## 2.5 Scheduling Distance Constrained Task Models

In the periodic model, the time windows for consecutive jobs $\tau_{i,j}$ and $\tau_{i,j+1}$ are predetermined but the actual times they get scheduled within these time windows are independent of each other. As illustrated in figure 2.2, $\mathrm{F}(\tau_{i,j+1}) - \mathrm{F}(\tau_{i,j})$ may be as little as $c_i$ and as much as $2p_i - c_i$; furthermore, as $c_i$ gets smaller, the maximum possible value tends to $2p_i$. An upper value which is twice as long as the one we had originally expected could clearly increase the job completion time jitter to unacceptably high levels. In addition, this much variation is simply inconsistent with the intuitive meaning of periodicity. If, for example, a backup system claims to run a periodic full backup (which takes 24 hours to complete)



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

once every week, then having two consecutive full backups 13 days apart would be a failure. One solution to this problem is to redefine the periodic task model with a more stringent deadline constraint so that $\mathrm{F}(\tau_{i,j+1}) - \mathrm{F}(\tau_{i,j})$ is bound by $p_i$. Specifically, if $\tau_{i,j}$ is finished by time $t^-$, then $\tau_{i,j+1}$ must finish within $[t \,..\, t+p_i)$; we would also require $\tau_{i,1}$ to complete by $[0, p_i)$. In scheduling literature, real-time task models with this scheduling constraint are known as **distance constrained** task systems [HL92], [HS95b], [HLH96a].

A distance constrained task, notated $\tau_{\mathrm{dc}} = (c, p)$ is represented by a **resource request** $c$ and a **distance constraint** $p$; again, we use subscripts, as in $\tau_{\mathrm{dc},i} = (c_i, p_i)$, when we deal with task sets[10]. Unlike periodic tasks, this model does not incorporate an explicit deadline parameter, possibly due to an already strong timing constraint; it is always assumed that the start time of a job is the implicit deadline of the previous job of the same task. In addition, tasks start at time $t = 0$; that is, an explicit $s_i$ parameter is missing as well.

The simplest way to schedule a distance constrained task set is to transform it into a periodic task set with tighter periods and apply periodic task scheduling techniques. For example, given a distance constrained task set $\mathrm{T_{dc}} = \{(c_i, p_i)\}$, a periodic version could be constructed by replacing each constraint with a period that is half in value: $\mathrm{T_p} = \{(c_i, \frac{p_i}{2})\}$. Assuming the latter is schedulable, the shorter periods guarantee that the distance constraints of the original task set are never violated. The cost of this simplicity, however, is that the tasks will now be scheduled twice as frequently as before (on average) and the achievable utilization will be halved: $\sum \frac{c_i}{p_i/2} = \frac{1}{2} \sum \frac{c_i}{p_i}$.

In [HL92] and [HLH96a], two algorithms based on specialization transform an arbitrary distance constrained task set into a harmonic form. In simple terms, **specialization**[11] is selectively shortening the distance constraints so that there is total alignment within the

[10]This is where the notation starts to get clumsy so we will avoid the "dc" subscript when the underlying task model is obvious from the context. Also, while variable $p_i$ is used as the *period* for periodic tasks, it is used as the *distance constraint* for distance constrained tasks. In some ways, this could be confusing... But since the two concepts are very similar, we preferred to overload an existing symbol rather than introduce a new one.

[11]Specialization was first defined in [HMR+89] in the context of pinwheel scheduling and later advanced in [CC92], [CC93] and others. However, we introduce it in conjunction with [HL92] and [HLH96a] since the order of presentation we follow in this survey (for the sake of what we hope to be a more flowing introduction) is different than the chronological order of the progress in real-time scheduling.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

task set; that is, after specialization, if one distance constraint $p_i$ is less than $p_k$, then $p_k$ is a multiple of $p_i$. In formal terms, given a base integer $q$ and a set of $n$ integers $\{p_0, p_1, \ldots, p_{n-1}\}$, the specialization function $\mathrm{S}(q, \{p_0, p_1, \ldots, p_{n-1}\})$ creates a new set $\{p'_0, p'_1, \ldots, p'_{n-1}\}$ where $p'_i$ is less than or equal to $p_i$ but greater than or equal to $q \times 2^k$ for the greatest $k$ possible. That is, $p'_i = q \times 2^k \mid \{(q \times 2^k \leq p_i) \wedge (\text{k is maximized})\}$. In the two examples of figures 2.5 and 2.6, we see that

$$\mathrm{S}(2, \{6, 9, 13, 18\}) = \{4, 8, 8, 16\}$$

$$\mathrm{S}(3, \{5, 11, 18, 19\}) = \{3, 6, 12, 12\}$$

Scheduler $S_c$ of [HL92] specializes the distance constraints of $\mathrm{T_{dc}}$ with respect to a single integer while scheduler $S_r$ of [HLH96a] considers a set of values to increase the chances of keeping the new task set utilization under one. The resulting harmonic task set is then scheduled with the rate monotonic priority assignment with the minimum distance constraint of $p_i - c_i$ for each task[12]. It is shown that as long as the utilization of the original distance constrained task set is less than or equal to $n(2^{\frac{1}{n}} - 1)$, then the utilization of the corresponding harmonic task set will be less than or equal to one. The importance of this result is that distance constraining tasks do not compromise schedulability, since $n(2^{\frac{1}{n}} - 1)$ is also the utilization bound for PC tasks. The use of the specialization and the rate monotonic priority assignments in the context of distance constrained tasks is referred to as **distance constraint monotonic scheduling** in [HLH96b].

The **pinwheel task** model[13] of [HMR+89] (which refers to distance constraints as **time**

[12]Unlike a maximum job completion time separation constraint, the distance constrained model does not specify a parameter to state a minimum separation constraint. Therefore, unless the scheduling algorithm maintains some notion of fair progress, a high priority task has the potential to dominate the use of the shared resource. This in fact turned out to be a problem in the construction of our network scheduler. As outlined in chapter 4, we ended up building a "braking" mechanism to prevent the thrashing of the shared resource (to borrow an operating systems term) when the combined requests of the periodic streams were low.

[13]Since its initial proposal in the context of processing satellite-based communications [HMR+89], the decision and the scheduling problems for the pinwheel model has been studied in [HRTV92], [CC92], [CC93], [RR97], and [LL97]. Its applications have included imposing end-to-end timing constraints in distributed systems [HLF95], real-time networks [HS95a] and regulating scheduling fairness [BL98]. Furthermore, it has been used as a tool for the scheduling of other types of distance constrained task systems in [HL92] and



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

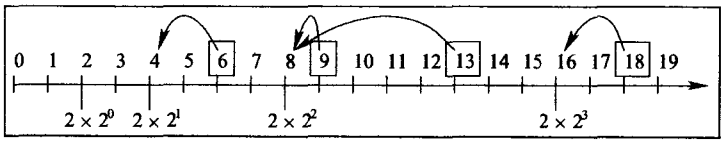



Figure 2.5: SPECIALIZATION WITH RESPECT TO A BASE VALUE OF 2: *In this example, we start with the task set* $\langle 6, 9, 13, 19\rangle$ *which has a density of* $\frac{1}{6} + \frac{1}{9} + \frac{1}{13} + \frac{1}{18} = \frac{16}{39} \approxeq 0.41$. *Applying the specialization process with a base value of 2 creates the task set* $\langle 4, 8, 8, 16\rangle$*; that is specialization leads to the distance constraint mappings of* $6 \rightarrow 4$, $9 \rightarrow 8$, $13 \rightarrow 8$, $18 \rightarrow 16$. *The density of this new task set is* $\frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{16} = \frac{9}{16} = 0.5625$ *which is higher than that of the original task set. However, since it is less than 1, the specialized task set is still schedulable. In addition, since the distance constraints are aligned without exception, the construction of a schedule after specialization is trivial.*

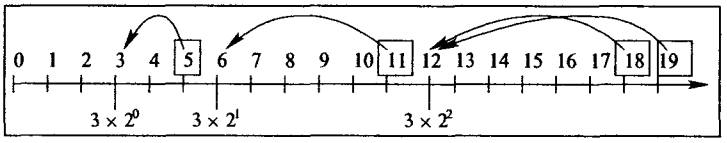



Figure 2.6: SPECIALIZATION WITH RESPECT TO A BASE VALUE OF 3: *In this example, the original task set is* $\langle 5, 11, 18, 19\rangle$ *with a density of* $\frac{1}{5} + \frac{1}{11} + \frac{1}{18} + \frac{1}{19} \approxeq 0.399$. *After specialization with respect to 3, we have a new task set* $\langle 3, 6, 12, 12\rangle$ *of density* $\frac{1}{3} + \frac{1}{6} + \frac{1}{12} + \frac{1}{12} \approxeq 0.667$. *Although the density increase is greater than the one in figure 2.5, the specialized task set is still schedulable.*

**windows**) is a special case of distance constrained scheduling where the resource requests are all set to one; that is

$$\mathrm{T_{pw}} = \{(1, p_0), (1, p_1), ..., (1, p_{n-1})\} = \{(1, p_i)\} \text{ for } 0 \leq i < n$$

Since all resource requests are one, a more convenient representation is simply listing the distance constraints in ascending order:

$$\mathrm{T_{pw}} = \{p_0, p_1, ..., p_{n-1}\} = \{p_i\} \text{ for } 0 \leq i < n$$

The associated scheduling requirement is that task $\tau_{\mathrm{pw,i}}$ should have the shared resource for *at least* one time unit out of every interval $[t \,..\, t + p_i)$, $t \geq 0$. For example if $\mathrm{T_{pw}} = \{4, 2, 4\}$, then repetitions of "$\tau_1, \tau_0, \tau_1, \tau_2, ...$" is a valid schedule. As a second example, if $\mathrm{T_{pw}} = \{2, 3\}$,

[HLH96a].



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

then repetitions of "$\tau_0, \tau_1, ...$" or "$\tau_0, \tau_0, \tau_1, ...$" are both valid schedules. With unit resource requirements, the issue of preemption is naturally eliminated. It is interesting to note that the formation of the pinwheel model has also allowed a number of prior stand-alone task models to be retrofitted into a larger framework. For example, if the scheduling condition is further constrained so that task $\tau_i$ should have the shared resource for *exactly* one time unit (as opposed to *at least* one time unit) out of every interval $[t \;..\; t + p_i)$, then the resulting model is equivalent to what has been known as the **periodic maintenance** problem [WL83].

The asymptotic complexity of determining whether a pinwheel task set is schedulable (and, if so, creating a schedule) is exponential in general. However, many efficient scheduling algorithms have been discovered for specific *subclasses* of the model. Typically, these subsclasses are defined by task set density and the number of unique time windows:

- The **density** of a pinwheel task is the reciprocal of its time window: $\mathrm{D}(\tau_{\mathrm{pw,i}}) = \frac{1}{p_i}$; unlike the utilization measure of a periodic task which is an *upper bound* on resource requests, the density measure is a *lower bound.* The density of a pinwheel task set is the sum of the densities of the member tasks: $\mathrm{D}(\mathrm{T_{pw}}) = \sum \mathrm{D}(\tau_{\mathrm{pw,i}})$. For **dense** sets, $\sum \frac{c_i}{p_i} = 1$ while for **non-dense** sets, $\sum \frac{c_i}{p_i} < 1$.

- Despite our use of set notation to represent the time windows, repetition is naturally allowed by the pinwheel model; it is perfectly reasonable for two or more tasks to have the same distance constraint[14]. The **number of unique time windows** is simply the cardinality of the task set once the repetitions are eliminated. For example, $\{x, y, x, x, z, y, z\}$ has 7 time windows but the number of *unique* ones is 3.

Figure 2.7 lists some of these subclasses that have been defined and studied in the literature.

In [HMR+89], it is shown that *harmonic* task sets can always be scheduled with a

[14]Perhaps we could have used a slightly different notation (as in $\mathrm{T_{pw}} = \langle p_0, p_1, ..., p_{n-1} \rangle$ instead of $\{p_0, p_1, ..., p_{n-1}\}$) to be consistent with set semantics but we preferred to keep all notational novelties to a minimum even if it meant that we had to make occasional clarifications in footnotes.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

| Name | Form | Description |
|---|---|---|
| $C_d$ | $\sum \frac{1}{p_i} = 1$ | Dense sets |
| $C_{0.5}$ | $\sum \frac{1}{p_i} \leq 0.5$ | Sets whose density are at most 0.5 |
| $C_h$ | $p_i < p_j \Rightarrow p_j \vert p_i$ | Sets with harmonic time windows |
| $C_{1,2}$ | $p_i \in \{i, j\}$ | Sets with two unique time windows |
| $C_{1,2,3}$ | $p_i \in \{i, j, k\}$ | Sets with three unique time windows |
| $C_{d,1,2}$ | $C_{1,2} \cap C_d$ | Dense sets with two unique time windows |
| $C_{d,1,2,3}$ | $C_{1,2,3} \cap C_d$ | Dense sets with three unique time windows |

Figure 2.7: PINWHEEL TASK SET SUBCLASSES: *The exponential complexity of schedulability for pinwheel task sets have been beaten in special subclasses. Here were enumerate seven of the ones encountered (and tackled) in scheduling literature.*

simple greedy algorithm as long as their density is less than or equal to one. This suggests that if *ordinary* task sets can be converted to harmonic versions in a way that would not create any deadline problems, then the actual schedule creation phase would be trivial. The specialization process we saw earlier in the context of distance constrained scheduling is the basis of this process as well; since the only modification is the shortening of the time windows, it is trivial to see that the new task set will satisfy the temporal constraints of the old one (i.e. the tasks are getting the resource either at the rate they wanted or more often). As before, the crucial element in specialization is finding a good base value $q$ so that the density increase is minimized. The $S_2$, $S_a$, and $S_x$ are three of numerous algorithms designed for this purpose ([HMR+89], [CC92], and [CC93]):

Scheduler $S_2$: Given $\mathrm{T_{pw}}$, creates a schedule for $\mathrm{T_{pw}^{'}} = \mathrm{S}(2, \mathrm{T_{pw}})$. Because $p_i' \in \mathrm{T_{pw}^{'}}$ is always greater than $\frac{p_i}{2}$, $\mathrm{D}(\mathrm{T_{pw}^{'}})$ is always less than $2 \times \mathrm{D}(\mathrm{T_{pw}})$. Consequently, members of the subclass $C_{0.5}$ can *always* be scheduled with $S_2$. This result is confirmed in [HL92] using a different approach.

Scheduler $S_a$: Given $\mathrm{T_{pw}}$, creates a schedule for $\mathrm{S}(p_0, \mathrm{T_{pw}})$; that is, the specialization is based on the smallest time window. $S_a$ is expected to perform better than $S_2$ since the time window whose modification is likely to cause the greatest increase in the task set (i.e. $p_0$) remains unchanged (please remember that the time windows are in ascending order).



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Scheduler $S_x$: Given $\mathrm{T_{pw}}$, creates a schedule for $\mathrm{S}(x, \mathrm{T_{pw}}) \mid (x \in [2 \,..\, p_0])$. Since the range[15] $[2 \,..\, p_0]$ covers the two extremes $\{2, p_0\}$, the schedulability bound of $S_x$ is at least as high as (and typically higher than) those of $S_2$ and $S_a$.

In [HMR+89], the $C_{d,1,2}$ subclass is proved to be schedulable. In [HRTV92], this result is generalized for $C_{1,2}$ which is a proper superset of $C_{d,1,2}$; that is, any pinwheel task set with only two unique time windows can be scheduled, regardless of whether the task set is dense or not. The schedulability of the $C_{1,2}$ class is utilized in [CC92] and [CC93] to create a new set of pinwheel schedulers based on the specialization operation. The basic idea is to first schedule two time windows $\{q_1, q_2\}$ only. The actual task set is also broken into two mutually exclusive sets: $\mathrm{T_{pw}} = \mathrm{T_{pw,q1}} \cup \mathrm{T_{pw,q2}}$. Subset $\mathrm{T_{pw,q1}}$ is specialized with respect to $q_1$ and scheduled in the time-slots allocated for time window $q_1$; similarly, subset $\mathrm{T_{pw,q2}}$ is specialized with respect to $q_2$ and scheduled in the time-slots allocated for time window $q_2$. The advantage of using two specialization operations as opposed to one is that each time window $p_i$ is aligned either with $q_1 \times 2^k$ or with $q_2 \times 2^k$ depending on which one creates a smaller increase in density.

The density based utilization bounds of seven well-known specialization methods are summarized in figures 2.8(a) and 2.8(b). It is also interesting to note ranking these algorithms purely on a schedulability threshold measure is misleading since, the infinite set of tasks that they can individually schedule is not always properly contained within one another. The Venn-diagram representation in figure 2.9 (which is from [CC92]) is a good indicator of this.

In [HL92] and [HLH96a], a general distance constrained task set $\mathrm{T_{dc}} = \{(c_i, p_i)\}$ is first converted into a pinwheel form $\mathrm{T_{pw}} = \{(1, \lfloor \frac{p_i}{c_i} \})$ and scheduled with $S_{x,y}$. The interpretation is that every $c_i$ slots allocated to $\tau_{\mathrm{pw,i}}$ adds up to one job of the original task $\tau_{\mathrm{dc,i}}$. The use of the floor function in the $\mathrm{T_{dc}} \rightarrow \mathrm{T_{pw}}$ transformation potentially increases the task density. Therefore, given the 0.7 density threshold of $S_x$, $\mathrm{T_{dc}}$ can be scheduled if $\sum_{i=0}^{n-1} \lfloor \frac{p_i}{c_i} \rfloor \leq 0.7$. From a practical point of view, the resulting system has more scheduling overhead as $\tau_{i,j}^{dc}$

[15]The actual range used in [CC93] is $(\frac{p_0}{2}..p_0]$ which is proved to be effectively the same as $[2 \,..\, p_0]$.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

| S | q | Bound |
|---|---|---|
| $S_2$ | $q = 2$ | $\frac{1}{2}$ |
| $S_a$ | $q = p_0$ | $\frac{1}{2}$ |
| $S_x$ | $q \in (\frac{p_0}{2} .. p_0]$ | $\frac{12}{20} = 0.65$ |

(a) Single parameter

| S | $q_1$ | $q_2$ | Bound |
|---|---|---|---|
| $S_{2,3}$ | $q_1 = 2$ | $q_2 = 3$ | $\frac{7}{12} = 0.58\bar{3}$ |
| $S_{b,c}$ | $q_1 = a$ | $q_2 \in \{\lfloor\sqrt{2a}\rfloor, \lceil\sqrt{2a}\rceil\}$ | $\frac{2}{3} = 0.\bar{6}$ |
| $S_{b,y}$ | $q_1 = a$ | $q_2 \in [a .. 2a)$ | 0.6964 |
| $S_{x,y}$ | $q_1 \in (\frac{a}{2} .. a]$ | $q_2 \in (a .. 2x]$ | 0.7 |

(b) Double parameters

Figure 2.8: SUMMARY OF PINWHEEL SPECIALIZATION ALGORITHMS: *The bulk of the work of proposing a new specialization operation is establishing a density threshold and showing that any task sets whose density is less than or equal to this threshold can be specialized with the proposed algorithm to a harmonic set whose density is at or below 1 (which guarantees schedulability). Part 2.8(a) summarizes three of the single parameter specialization processes while part 2.8(b) summarizes four of the double parameter ones.*

will be preempted $c_i$ times.

Finally, a **general pinwheel** task model eliminates the $c_i = 1$ rule and thus the scheduling constraint requires the member tasks to have the shared resource for at least $c_i$ time units out of every interval $[t .. t + p_i)$, $t \geq 0$. Simple mapping operations are then used to transform a restricted pinwheel task set into a general pinwheel task set and vica versa. The generalization of the pinwheel model is relevant for two reasons. First, the pinwheel model can be applied to a wider set of scheduling problems. Second, the generalization makes a sufficient difference in the length of the input task set representation to positively effect the asymptotic complexity of schedulability tests [BL98].

## 2.6 Dynamic Scheduling

As discussed earlier, every job of a task in static scheduling is assigned the same priority; different jobs in dynamic scheduling, however, may be assigned different priorities based on



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

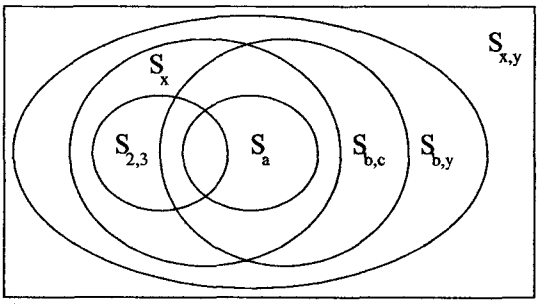



Figure 2.9: PINWHEEL SPECIALIZATION ALGORITHMS: *This diagram compares the "strengths" of specialization algorithms $S_2$, $S_a$, $S_x$, $S_{2,3}$, $S_{b,c}$, $S_{b,y}$. We can see that the set of task sets schedulable by $S_{by}$ is a proper subset of the set of tasks schedulable by $S_{xy}$; that is $S_{xy}$ can schedule every pinwheel task set that can be scheduled by $S_{by}$ but the converse is not necessarily true. However, although there is an overlap between the sets schedulable by $S_{23}$ and $S_a$, neither properly contains the other. Scheduler $S_{x,y}$ turns out to be the best one: it has the highest density bound and its set of schedulable pinwheel task sets properly contains any other task set that can be scheduled by the other six algorithms.*

the instantaneous conditions of the task set and the underlying system. Due to its flexibility, dynamic scheduling can potentially achieve higher levels of utilization than static scheduling. However, it is also associated with a number of disadvantages. First, dynamic scheduling is not as predictable as static scheduling. Second, it is more complicated to implement efficiently and it may lead to higher levels of scheduling overhead. Third, for absolute hard real-time systems, the idea of utilizing run-time conditions is rejected as a basis for scheduling; the very consideration of conditions beyond $t = 0$ implies a lack of predictability which cannot be tolerated due to the extreme cost of missing a deadline. Finally (and most importantly for our project), dynamic scheduling requires instantaneous knowledge of the tasks in order to achieve its full potential. As will be outlined in chapter 4, this is not possible in network scheduling.

Since our network protocol is not based on dynamic scheduling, the focus of this survey is mostly on static scheduling. However, for completeness, we still enumerate a number of well-known results on dynamic scheduling in case the reader would like to pursue this branch of the field:

- In [LL73], it is shown that a necessary and sufficient condition for the schedulability



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of a PC task set is $\sum \frac{c_i}{p_i} \leq 1$. This result is extended in [LW82].

- [RS84] and [SR85] are two of the early papers that establish some of the earlier results on dynamic scheduling.
- The work of [CC89] is one of the most frequently cited in dynamic scheduling papers.
- **Pfair scheduling** is a scheme that has a timing constraint even stronger than that of distance constrained scheduling. We chose not to include it in our presentation since it seemed inapplicable to network scheduling. However, for the record, it has a static as well as a dynamic version, the difference of which is spelled out in [Bar95].
- The "sliver" method in [HL92] and [HLH96a] is an example of how dynamic priority schemes could be applied to the distance constrained model. As cited earlier, [HMR+89] shows that a pinwheel task system is always schedulable if its density is at most $\frac{1}{2}$; the same conclusion can be reached by using the sliver method to schedule distance constrained task sets.

## 2.7 Resource Request Variations

Given the wide range of systems where real-time scheduling algorithms could be applied, modeling the resource requirements of $\tau_i$ by a single constant $c_i$ is not sufficiently realistic. Typically, the dynamic resource requirement $\tau_i(i,j)$ of $\tau_i$ varies from job to job and its average request per instance is lower than $c_i$. The following are some of the approaches that have been investigated to deal with variable resource requests. In order to eliminate ambiguity, task models based on the assumption of constant (or worst case) resource requests will be referred to as **regular** tasks while those that use the functional notation of $C(i,j)$ will be referred to as **irregular** tasks [MBCG98].

For some systems, $C(i,j)$ follows a predetermined pattern. For example, MPEG compressed video streams can be modeled as the repetitions of fixed sequences of I, P, and B type frames (e.g. "I-B-B-P-B-B" repeated) where each frame type implies a different



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

level of bandwidth request ([Gal91], [PeZ94], [HK95]). The **multi-frame** task model of [MC96b] is proposed for such systems. Specifically, the resource requirements of a task is represented with a sequence of $m \geq 1$ fixed values $c_i^* = (c_i^0, c_i^1, c_i^2, \ .. \ c_i^{m-1})$, such that $C(i,j) = c_i^{(j \bmod m)}$. A multi-frame task is then represented as a set of pairs $T_{mf} = \{(c_i^*, p_i)\}$ where $p_i$ is the separation[16] between consecutive jobs of task $\tau_{mf,i}$. It is shown in [MC96b] that if a multi-frame task set posseses a property referred to as **accumulative monotonicity**, then an instant can be identified with each $\tau_{mf,i}$ when it submits the heaviest scheduling load. This makes it possible to implement a feasibility test analogous to the critical instant test and to define a utilization based schedulability bound. A utilization based schedulability bound is defined by quantifying the variation in $C(i,j)$. For each $\tau_{mf,i}$, $r_i$ represents the ratio of $\tau_{mf,i}$'s maximum request and the request immediately following the maximum request in the $c_i^* = (c_i^0, c_i^1, c_i^2, \ .. \ c_i^{m-1})$ sequence. For $r = \min\{r_i\}$, a multi-frame task $T_{mf}$ is schedulable if its utilization with respect to its maximum resource request is below $r \times n((\frac{r+1}{r})^{1/n} - 1)$. When $r = 1$, this expression reduces to $1 \times n((\frac{2}{1})^{1/n} - 1) = n(2^{1/n} - 1)$ which is the utilization bound for the PC type periodic task model.

In many cases, it is unlikely (or even impossible) that two consecutive jobs of $\tau_i$ will each request $c_i$ units of the shared resource. Therefore, if every pair of jobs $\tau_{i,2j}$ and $\tau_{i,2j+1}$ for $j \geq 0$ can be aggregated (i.e. merged) into a single "larger" job, then the maximum resource request of each of these new jobs will be less than $2c_i$. Generalizing this observation from 2 to $k$, every $k$ consecutive jobs $\tau_{i,kj}, \tau_{i,kj+1}, \tau_{i,kj+1}, \ .. \ \tau_{i,kj+(k-1)}$ of the original task can be aggregated into a single new job whose total resource usage will be less than $kc_i$. Under the simplicity of the PC model, the period and the deadline of the aggregated jobs will be $kp_i$. However, an accurate estimate of the maximum resource usage of $k$ jobs requires additional assumptions to be made about the scheduled task system. Increasing the value of $k$ absorbs the effects of the infrequent peaks in $C(i,j)$ and the resource requests of the aggregated jobs of $\tau_i$ tend toward $k$ times the average resource request of $\tau_i$. Of course, aggregation

[16] In [MC96b] $p_i$ is a *minimum* separation which makes the multi-frame model an extension of sporadic tasks. However, when proving schedulability issues, the separation is assumed to be a constant as in the PC model.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is possible only if deadlines can be postponed. In [Leh90], it is shown that the required analysis is considerably simplified if deadlines are postponed in multiples of task periods.

In the **general task model** of [MC96a], each task $\tau_i$ is associated with a set $\sigma_i^* = \{\sigma_i^0, \sigma_i^1, \sigma_i^2, \ldots\}$ where $\sigma_i^j$ represents the maximum total resource requirement of $j+1$ jobs of $\tau_i$. A general task set is defined as $\mathrm{T_g} = \{(\sigma_i^*, p_i)\}$ where $p_i$ is the minimum (i.e. sporadic) or exact (i.e. PC periodic) separation between the jobs of $\tau_i$. The consideration of sums of resource requirements eliminates the need for accumulative monotonicity. A mapping process $\tau_{\mathrm{g,i}} \rightarrow \tau_{\mathrm{mf,i}}$ using the relation $c_i^* = (\sigma_i^0, \sigma_i^1 - \sigma_i^0, \sigma_i^2 - \sigma_i^1, \ldots, \sigma_i^{m-1} - \sigma_i^{m-2})$ is given to transform general tasks into multi-frame tasks so that the analysis techniques developed for multi-frame tasks can also be applied to general tasks. Another approach based on the aggregation of consecutive jobs is investigated in [MBCG98] with the **enhanced frame model**.

A common feature of all of the regular and irregular task models given so far is that the resource requirements are assumed to be known. However, for some systems, a statistical description of the task requests may be more appropriate that an enumeration of predetermined values. This is the approach taken in [AB98] where the PC periodic model is enhanced with the use of probability distribution functions to specify the resource requirements of tasks. Resource requirement variations between consecutive task jobs are smoothened by considering resource utilizations within **superperiods**. Assuming tasks are sorted by their period lengths, the superperiod of a $\tau_i$ is the period of the next task $p_{i+1}$. Therefore, the superperiod of $\tau_i$ is $\frac{p_{i+1}}{p_i}$ times longer than its period. The schedulability criteria is that each task needs to meet sufficient deadlines in its superperiod to satisfy a "quality of service" condition. This approach is unique with its relaxed feasibility condition since the definition of feasibility for most other schedules has been "meeting all the deadlines all the time".



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

## 2.8 Conclusion

In this chapter, we provided a survey on static scheduling algorithms. Given the extensive research of the past three decades, there are many additional topics that could have been included (e.g. temporary overload management[KS95], value based arbitration of resource requests [BSS95], dual-priority assignment [DW95], etc) but our coverage was intentionally limited to those algorithms that have the potential to be used in the network scheduling problem. In the remaining portion of this writeup, we focus on the rate monotonic scheduling and the distance constrained algorithms for the development of a network protocol.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# Chapter 3

# Schedulability Simulations

The evaluation of a real-time scheduling algorithm consists of two stages. First, the schedulability characteristics of the algorithm are defined through mathematical models or simulations. Simply put, this is the effort to quantify the percentage of task sets that can be scheduled by the given algorithm and thus relates to the real-time definition of "optimality" as discussed in chapter 2. The second stage is the investigation of the characteristic of the schedules produced by the algorithm. In this chapter, we explore the schedulability of Rate Monotonic Scheduling (RMS) and Distance Constrained Scheduling (DCS); their schedule characteristics are discussed in chapters 5 and 6. Of all the algorithms covered in chapter 2, these two appear to be good candidates for driving a real-time network protocol; they are static scheduling algorithms, their preemption task switch rates are not excessive, etc. An additional goal is to better understand the effects of non-zero task switch times. As mentioned in the introductory chapter, this is one of the fundamental differences between processor scheduling and network scheduling and, as we show in this chapter, has far reaching effects.

We start with a description of how random task sets are generated; this information is relevant for chapters 3, 5, and 6 where we present simulation results. We then discuss the schedulability characteristics of RMS (section 3.2) and DCS (section 3.3). Finally, we compare the performance of the two algorithms (section 3.4) before concluding the chapter.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

## 3.1 Generation of Random Task Sets

The schedulability experiments of this chapter as well as the performance experiments of 5 and 6 are based on five attributes that determine the characteristics of the randomly generated task sets:

- Number of tasks, $n$.
- Total task set utilization, $U$. For a given task set, this is the sum of the $\frac{c_i}{p_i}$ values of member tasks. For rate monotonic tasks, the $p_i$ values represent periods while for distance constrained tasks, $p_i$ values represent distance constraints. The range of $U$ is between 0 and 1.
- Period lengths. In order to express task sets in a number-independent manner, we keep track of the $\frac{\text{shortest period length}}{\text{longest period length}}$ ratio $R$.
- The resource-request to period-length correlation. For example, when this value is positive, a task with a large utilization is more likely to have a long period.
- Task switch overhead.

In each simulation, the generation of a random task set is driven by a target total utilization value $U$. For a set of $n$ tasks, $n-1$ uniformly distributed random variates are first drawn from the interval $(0 \rightarrow U)$. The $n$ gaps between 0, the random values, and $U$ are then used as the $n$ individual task utilizations $u_i$. This way, as illustrated in figure 3.1, we maintain the $\sum u_i = U$ invariant even though the individual task utilizations are random. Generating $n$ task periods, on the other hand, follows one of two schemes

1. As shown in figure 3.2(a), the system simply draws $n$ random period values $p_i$ between the user specified period-minimum/maximum limits using a uniform distribution.
2. As shown in figure 3.2(b), the system draws two sets of values. First, a set of integers containing the powers-of-two that would fall within the period-minimum and period-maximum interval are drawn. For example, if the period range is 64 to 511, then



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

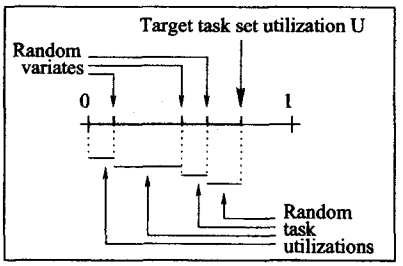



Figure 3.1: GENERATION OF TASK UTILITIES: *The gray area represents the interval from which the uniformly distributed random variates are drawn. The gray area in part 3.2(a) represents the valid range for the task periods.*

this set of integers contains {6, 7, 8}. Given a randomly chosen exponent $p$ from this set, the system draws a second random number from the interval $[2^p \rightarrow 2^{p+1})$. The indirect uniform distribution makes it equally likely for each power-of-two interval (shown in increasingly darker shades of gray in figure 3.2(b)) to contain the period of a particular task so that the behavior of algorithms that work by specialization can be better studied. As defined in the previous chapter, specialization is the process of shortening task periods such that a particular period $p_i$ becomes an even multiple of every period shorter than $p_i$. Although different specialization techniques lead to different period modifications, all of them make use of powers of two to impose a total ordering based on the "even-multiple" principle.

When we need to specify which of the above two mechanisms is used in task set generation, we refer to the second one as the **indirect uniform distribution** for period selection. Once $n$ task-period/utilization pairs $(p_i, u_i)$ are computed randomly, the simulation computes the resource requests $c_i$ using the equality $u_i = \frac{c_i}{p_i}$.

## 3.2 Rate Monotonic Scheduling

It was shown in [LL73] that rate monotonic priority assignments guarantee schedulability as long as the total utilization of $n$ tasks remain at or under $n(2^{1/n} - 1)$. Since this condition is *sufficient* but not *necessary*, it is also known that many task sets with total utilizations



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

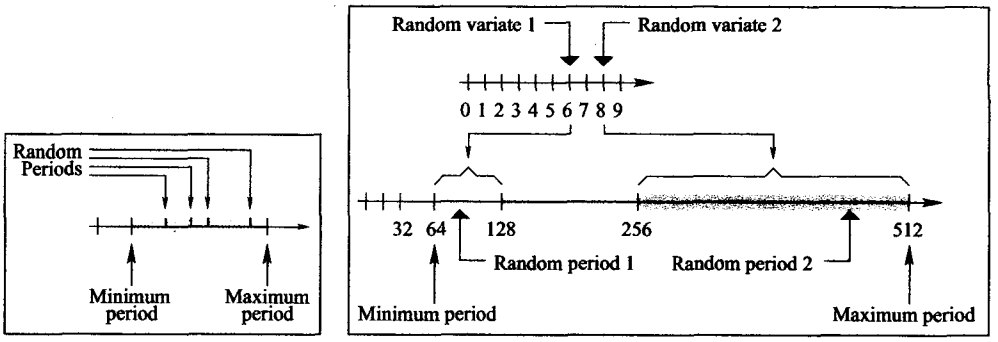



Figure 3.2: GENERATION OF TASK PERIODS: *In part 3.2(a), we see that the system draws the periods of the task set from the interval set by the user. In order to test the performance of scheduling algorithms based on specialization, the simulator optionally uses a two level random number generation when picking task periods. As shown in part 3.2(b) The interval from which the periods are to be drawn is broken into subintervals partitioned by powers of two (shown in different shades of gray in the figure). The first random number chooses one of the subintervals and the second chooses a uniformly distributed random value from this subinterval.*

exceeding this limit are schedulable as well [LSD89]. For this reason, our first sequence of simulations are designed to better understand the performance of RMS when the total task set utilization is within the gray area of $n(2^{\frac{1}{n}} - 1)$ to 1.

Figure 3.7 shows eight plots to gauge the performance of RMS; no correlation exists between the requests and periods of the tasks of these plots. As expected, the utilization based lower bound for a given number of tasks is in general lower than the actual utilization levels at which we start seeing temporal failures. This is because the lower bound is based on the worst possible task that one could construct, in an adversarial way, against the operation of the algorithm. Even though we generate a large number of task sets for each plot point, it is very unlikely for us to encounter this one particular task set (or any that is close to it in "badness") by chance. We also see in figure 3.7 that the schedulability spread for different $\frac{\text{longest task period}}{\text{shortest task period}}$ ratios are effected by two parameters. The spread increases when indirect uniform distribution is used for picking task periods. This spread also increases when the number of tasks increase. As a consequence, the further we are from the conditions that



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

are shown in the top left plot (i.e. direct uniform distribution is used for periods and the number of tasks are low), the greater is the underestimation of the lower bound test that determines schedulability based on utilization alone.

Figure 3.8 shows a different slice through the same set of simulation results. Schedulability is still plotted as a function of utilization, but (i) the separate curves in each plot now represent task sets of different sizes, and (ii) the values of the $\frac{\text{longest task period}}{\text{shortest task period}}$ ratio now increase as we go from the top row down. Consequently, we are looking at the data that originally lead to the plots in 3.7, but with the hierarchical order of the "number of tasks" and $\frac{\text{longest task period}}{\text{shortest task period}}$ parameters having swapped places.

In chapter 1, we stated that one of the main differences between the processor and the network scheduling problems is the magnitude of the scheduling overhead, especially in terms of task switching. With processor scheduling, each task (including the scheduler) is a process on the processor and there is often hardware support to reduce the overhead of multitasking. Therefore, the cost of typical scheduling actions such as detecting the completion of a job, sensing the admission request of a new task, switching from one task to another, etc, is often negligible. In some case, the context of networking does not immediately change this picture. For example, one application of the Statistical RMS algorithm [AB98] is to coordinate the scheduling of multimedia generated cells in an ATM switch, but this is practically identical to the processor scheduling problem. Non-zero overhead becomes an issue when the processes of a real-time system (i.e. the controller process as well as the competing tasks) are distributed (as in the case of the hosts of a LAN) that leads to communication costs that must be worked into the operational overhead of the protocol. This is the case in our project. Therefore, before we implement a network protocol based on processor scheduling algorithms, we need to throughly understand the schedulability consequences of this particular and significant difference between the two models. The magnitude of the task switch cost is the most important one of all.

One of the strengths of RMS is the efficiency of its schedulability test. As stated earlier, a set of $n$ tasks is immediately schedulable if its total utilization is at or below $n(2^{1/n}-1)$; if



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

not, the task set is still schedulable if it passes the critical region test which verifies that all the deadlines during the first job of the task with the longest period are met. Unfortunately, as we incorporate a non-zero task switch into the operation of this algorithm, the critical region test ceases to be a reliable schedulability indicator. This initially seems unexpected since the presence of overhead can be thought as being equivalent to increasing resource requests by the overhead amount; however, figure 3.3 shows the flaw in this logic. In part 3.3(a), we see the first 24 time slots of a RMS schedule for the task set $\{(2,6),(3,11)\}$. In part 3.3(b), we attempt to model the effect of a single time slot task switch overhead by increasing the resource requests by one slot and (thus) scheduling $\{(3,6),(4,11)\}$; we note that this increase does not create any deadline failures. In part 3.3(c), on the other hand where we try to schedule $\{(2,6),(3,11)\}$ with an overhead value of 1 (shown with the circles), the second job of the lower priority task misses its deadline (shown with the black triangle). Specifically, as the third job of the second task starts, the second job is not yet complete.

The problem stems from a number of reasons. First, the number of task switch overheads incurred by a job is a function of the number of times it gets interrupted by higher priority tasks. Therefore, if we want to fix the schedulability test by resource request modifications, they will have to be increased by multiples of the overhead, which will, in many case lead to the underutilization of the shared resource. Second, the idle gaps within the schedule that are less than or equal to the overhead duration are no longer usable; it is as though the small amounts of the shared resource leaks away without the knowledge of the scheduler. Third, overhead intervals are "uninterruptable" by nature and thus they disrupt the preemptive operation of the scheduler. And finally, although the greatest aggregate resource request impact still occurs at time $t = 0$, the greatest aggregate overhead impact does not occur within the first critical region interval of the schedule (this issue depends on the particular values of the periods found in the task set).

Figure 3.9 shows the results of the simulations that investigate the effect of the overhead on schedulability. First, we can see that as the size of the task set increases (which



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

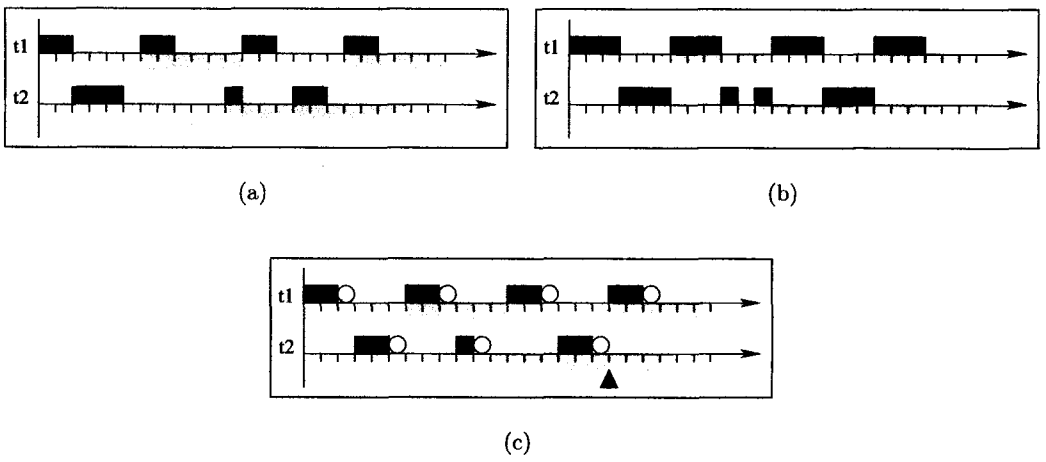



Figure 3.3: FAILURE OF THE RMS SCHEDULABILITY TEST WITH NON-ZERO TASK SWITCH OVERHEAD: *In each of the three timelines, the job completion durations are shown with rectangles of alternating color under the timeline arrows. The darker rectangles above the timeline arrows indicate when the resource request is granted to particular jobs.*

corresponds to looking at the plots further down), the failure rate increases as well. Second, we notice that the utilization-overhead pairs that lead to failures forms a band around the task sets that actually need the critical region test. That is, when the utilization is low, the ample availability of the resource absorbs the cost of the overhead while when the utilization is high, the task set is not schedulable even with a zero overhead. Third, as the number of tasks is increased, the critical region failure band shifts toward lower utilization levels. This is consistent with our earlier observation that, in general, the likelihood of schedulability is inversely proportional to the number of tasks in the set.

## 3.3 Distance Constrained Scheduling

As outlined in the survey chapter, distance constrained schedulers schedule the specialized form of the given task sets. We know that any specialized task set whose total utilization remains at or under 1.0 is schedulable. Therefore, the schedulability lower bound of DCS is based on deriving threshold expressions such that if the original task set density is lower than this threshold, then the density of the specialized task set is guaranteed not to exceed 1



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(and is thus guaranteed to be schedulable). In the case of the $S_r$ method from [wHL01], the specialization threshold turns out to be $n(2^{1/n}-1)$ which is identical to the feasibility lower-bound of RMS. However, the worst case limit is only one piece of a complete evaluation. For example, the authors of [wHL01] note that the $S_r$ based distance constrained algorithm produces schedules with less job completion time jitter than RMS. The tests we outline in this section show that there are additional differences between these two algorithms.

Figure 3.10 shows the distance constrained schedulability plots. Compared to the rate monotonic results (which were shown in figure 3.7), the $\frac{\text{longest task period}}{\text{shortest task period}}$ ratio as well as whether the uniform or indirect uniform distribution is used for period selection appear to have little or no effect on schedulability. In figure 3.11 where we swap the plot hierarchy of the "number of tasks" and "$\frac{\text{longest task period}}{\text{shortest task period}}$ ratio", we observe once more that the single most crucial schedulability factor for DCS is the number of tasks in the set. This is a consequence of specialization which is a many-to-one mapping from the set of all real-time task sets to *specialized* real-time task sets. Through this mapping, minor differences (as in different max to min period ratios) disappear and thus make no schedulability difference.

It is often the case that when a task set is specialized, subsets of tasks get mapped to the same shortened constraint. For example, if we have $T_{dc} = \{(1,5),(1,9),(3,10)\}$, then specializing with respect to a base value of 4 leads to $\{(1,4),(1,8),(3,8)\}$ and thus the second and the third tasks end up with the same distance constraint of 8. When the task switch cost is 0, there is no scheduling consequence to swapping the priorities of the tasks with equal distance constraints. For example, as we see in figure 3.4, scheduling $\{(1,4),(1,8),(3,8)\}$ is equivalent in a real-time sense to scheduling $\{(1,4),(3,8),(1,8)\}$ since neither leads to a deadline failure (though we do not have a formal proof, it is fairly straight forward to see that this example can be generalized to any schedulable task set under DCS). If the task switch has a non-zero cost, however, then the schedule we see in part 3.4(b) is better than the schedule 3.4(a) since the former requires fewer task switches per major cycle.

In figures 3.12 and 3.13, we see the simulation results that utilize this particular characteristic of DCS. In each of the surface plots, an individual point represents the percentage



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

improvement in schedulability by trying different priority orders within specialization levels. Specifically, for each utilization/overhead combination, we count the number of *unschedulable* tasks sets which turn out to be *schedulable* after the member tasks are sorted (within the specialization levels) either in ascending or in descending order by their request size ($x$). The gain in schedulability is then quantified (as a percentage figure) with $100 \times \frac{x}{1000}$ since we had 1000 task sets per point. We can see that depending on the characteristics of the task sets, this algorithmic modification has a variable level of effectiveness and range from hardly making any difference to about a 6% increase. Overall, the benefits are greater when the $\frac{\text{longest task period}}{\text{shortest task period}}$ ratio is low and when the number of tasks is high.

Having seen this improvement, it is tempting to look for a method that determines an ideal priority assignment order within specialization levels. Clearly, the goal is to find one arrangement that reduces the number of task switches for the entire task set. Unfortunately, it appears a greedy algorithm cannot be used for this purpose; that is, a globally optimal solution cannot be determined by considering priority assignment problems in isolation at each level. Due to time constraints, we did not attempt a proof to show that this problem is NP-complete but, based on intuition, it seems likely that the underlying search process will have exponential complexity. If so, a suboptimal approximation algorithm (e.g. those for the bin packing problem) may still pay off.

## 3.4 Rate Monotonic vs Distance Constrained Scheduling

At the beginning of this chapter, we stated that one of our goals is to understand the schedulability difference between RMS and DCS within the utilization/density interval of $n(2^{\frac{1}{n}} - 1)$ to 1 ($n$ being the number of tasks). The plots in figures 3.7 and 3.10 already showed one notable difference. Namely, while RMS usually performs *above* the worst case schedulability level, DCS remains *at* its worst case level regardless of the parameters of the task set; in order to simplify the comparison, figure 3.14 replicates some of the plots from figures 3.7 and 3.10 side by side. The crucial observation here is that, on average, RMS is



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

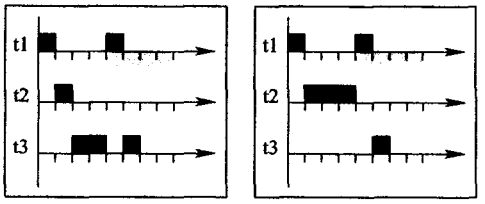



Figure 3.4: SCHEDULABILITY CONSEQUENCES OF SORTING TASKS: *We assume that we are given the task set* $T_{dc} = \{(1,5),(1,9),(3,10)\}$. *In part 3.4(a), we see the distance constrained schedule for* $S(4, T_{dc}) = \{(1,4),(1,8),(3,8)\}$. *In part 3.4(b), we see the schedule for the effectively equivalent specialized task set of* $\{(1,4),(3,8),(1,8)\}$. *With no task switching overhead, these two schedules are the same. With a non-zero task switch, however, the second one is better since it has one fewer task switch.*

the stronger of the two algorithms. The particular task set we see in figure 3.5 shows the cause of this difference. Even though $T_{rms} = \{(4,10),(7,15)\}$ is schedulable, the excessive shortening of the distance constraints by the $S_r$ scheme pushes the density of the specialized task set $T_{dc} = \{(4,10),(7,15)\}$ above 1.

In figure 3.16, we see the simulation results that show the consequence of this difference in a wide context, *when the task switch overhead is 0.* In each of the surface plots, an individual point represents the percentage difference in schedulability for the two algorithms. Specifically, for each utilization/overhead combination, we count the number of task sets that are schedulable by one algorithms, but not the other; this difference is then represented as a positive percentage value for a rate monotonic advantage and a negative percentage value for a distance constrained advantage. In almost every situation, RMS consistently performs better; in addition, the difference grows when the number of tasks are increased.

When, however, we increase the task switch overhead to values greater than 0, RMS loses its edge. A particular example of this can be seen in figure 3.5 where the distance constrained version of a task set in part 3.5(a) meets all its deadlines but the rate monotonic version in part 3.5(b) does not. The results of the simulations that explore the consequences of this issue in a wide context are shown in figure 3.16. It is important to note that, unlike the



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

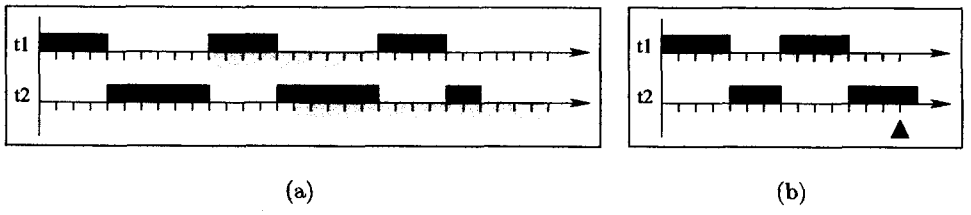



Figure 3.5: RMS EDGE OVER DCS: *As we see in part 3.5(a), the task set* $T_{rms} = \{(4, 10), (7, 15)\}$ *is schedulable. On the other hand, the distance constrained task set* $T_{dc} = \{(4, 10), (7, 15)\}$ *which specializes to* $T'_{dc} = \{(4, 7), (7, 14)\}$ *we see in part 3.5(b) by* $S_r$*, is not.*

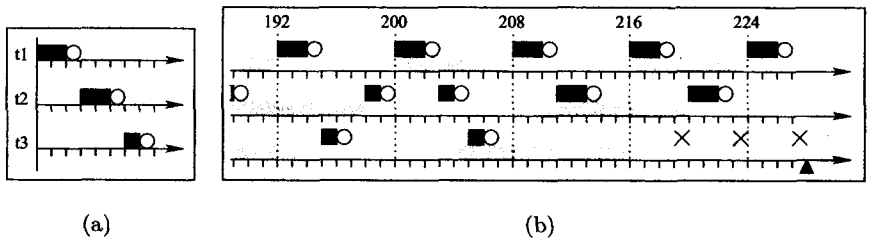



Figure 3.6: DISTANCE CONSTRAINED SCHEDULING EDGE OVER RATE MONOTONIC SCHEDULING: *As we see in part 3.6(a), the task set* $T_{dc} = \{(2, 8), (2, 11), (1, 12)\}$ *is schedulable when specialized to* $T_{dc} = \{(2, 8), (2, 8), (1, 8)\}$ *despite a task switch overhead value of 1 (overhead durations shown with circles).* $T_{rms} = \{(2, 8), (2, 11), (1, 12)\}$ *we see in part 3.6(b), on the other hand, is not; the job of the third task that starts at time* $t = 216$ *cannot find an interval long enough to run (the cross symbols indicates the presence of intervals that are too short to be useful).*

previous case, neither of the algorithms dominates. As before, the edge of RMS surfaces when the $\frac{\text{longest task period}}{\text{shortest task period}}$ ratio or the numbers of tasks increase (these are then regions where the surface plot is above the 0 level). The distances constrained scheduler, on the other hand, catches up as the ratio of the overhead to the average request size increases (these are then regions where the surface plot is below the 0 level).

## 3.5 Conclusion

As we wrap up this chapter, DCS appears to have a greater number of advantages than RMS for the design of a network protocol. First, due to the effect of specialization, the



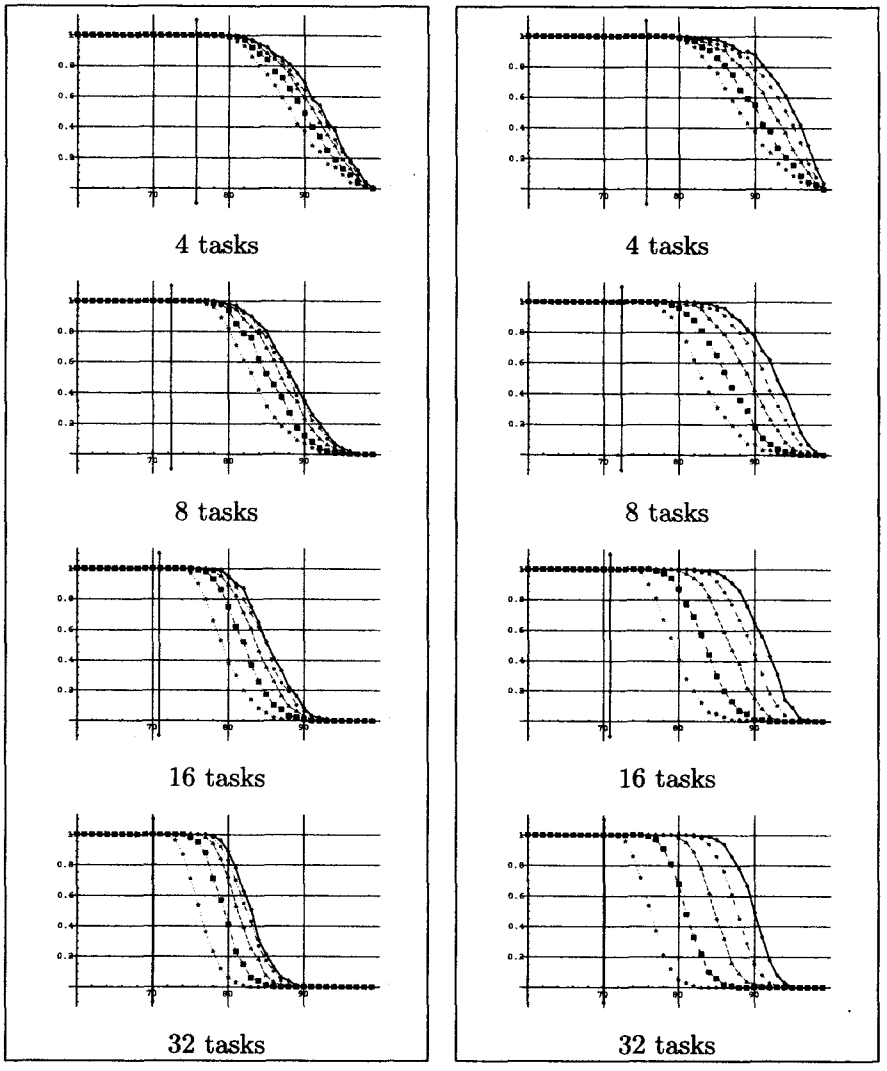
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

major cycle length of a distance constrained scheduler is much shorter than that of a rate monotonic scheduler; since our goal is to construct and store data structures to represent schedules, this translates into significant space saving. Second, by considering different priority assignments for the tasks that get mapped to the same specialization level, the schedulability of DCS is increased. Third, as the overhead value of the underlying system increases, DCS outperforms RMS. RMS, on the other hand, is consistently better than DCS when the task switch overhead is 0; unfortunately, this is a strength that is insignificant for network scheduling.

Our next goal is to compare the runtime characteristics of these two algorithms, especially in the face of changing system parameters. However, we first need to create a simulation environment as well as a network protocol for the required simulations. The details of these two steps are presented in the next chapter.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



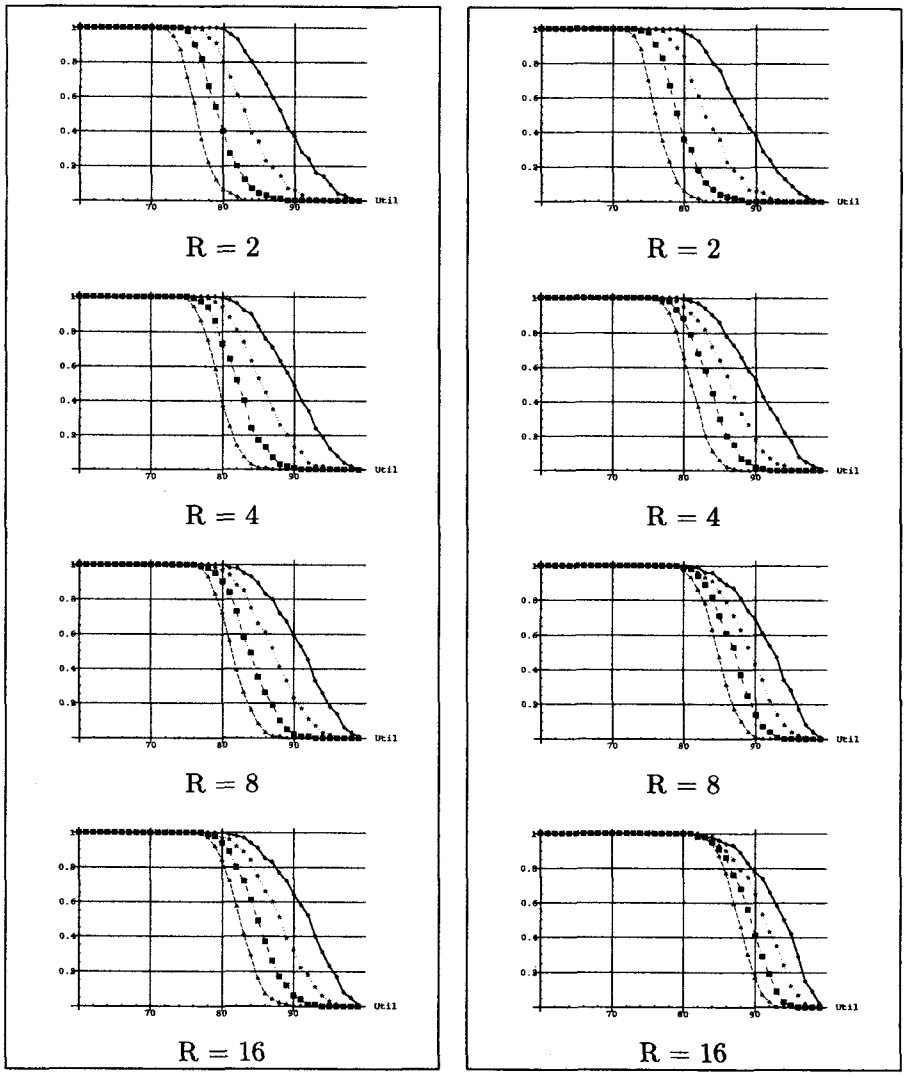

(a) Uniform distribution for random selection of periods

(b) Indirect uniform distribution for the random selection of periods

Figure 3.7: RATE MONOTONIC SCHEDULABILITY PLOTTED AS A FUNCTION OF UTILIZATION (PM CURVES): *The dependent variable of schedulability is a function of the total task set utilization. No correlation exists between the requests and periods of the tasks. Separate curves in each plot represent* $\frac{\textit{longest task period}}{\textit{shortest task period}}$ *ratios of 2, 4, 8, 16, and 32. The vertical line to the left of all the curves (connecting two diamonds) shows the utilization based lower bound for the given number of tasks. Each data point represents the fraction of 1000 randomly generated task sets that could be scheduled.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



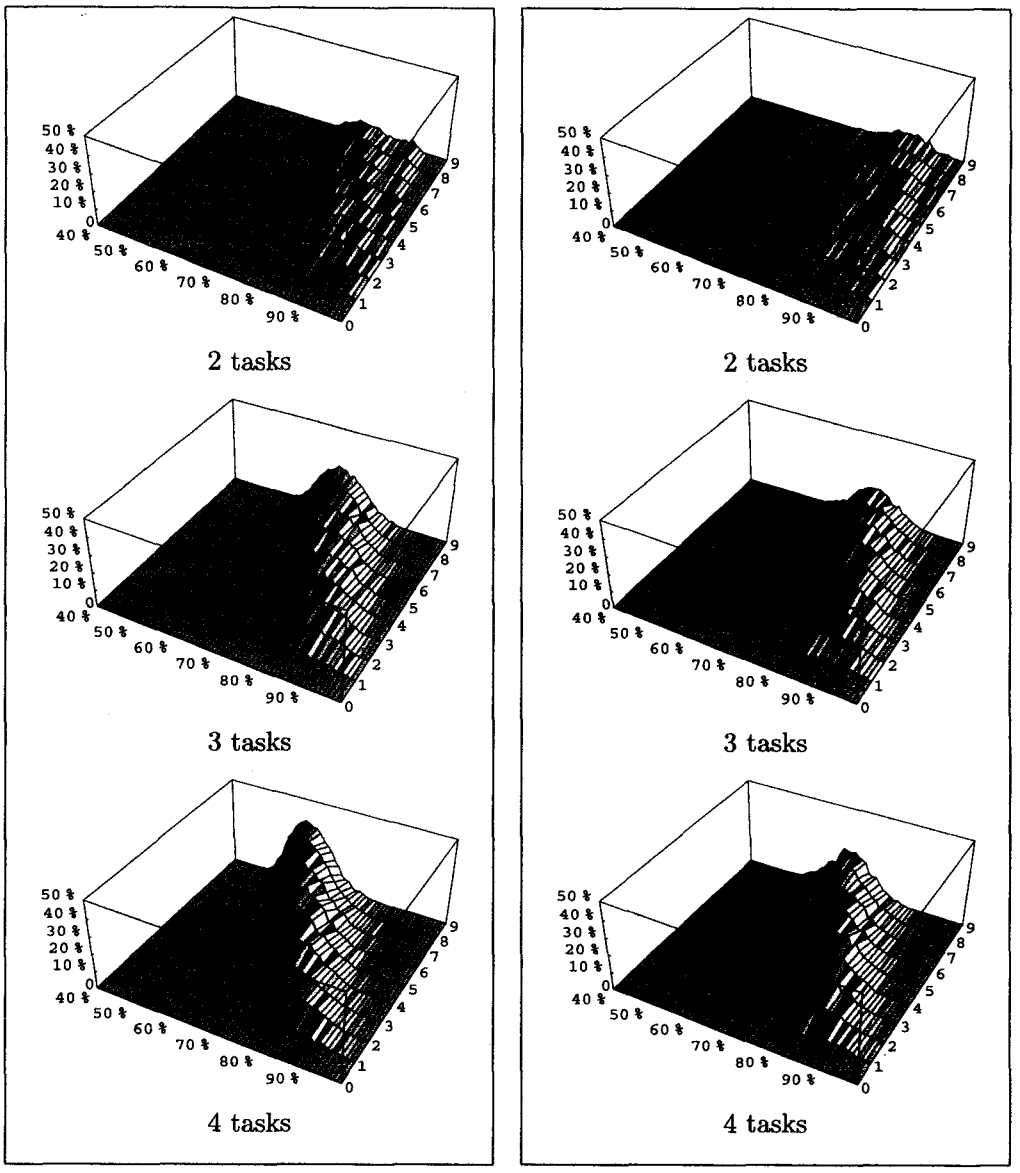

(a) Uniform distribution for the random selection of periods

(b) Indirect uniform distribution for the random selection of periods

Figure 3.8: RM SCHEDULABILITY PLOTTED AS A FUNCTION OF UTILIZATION (TSS CURVES): *These plots provide a different slice through the information shown in figure 3.7. Again, the dependent variable of schedulability is a function of the total task set utilization. No correlation exists between the requests and periods of the tasks. However, separate lines in each plot now represent different size task sets. Each data point represents the fraction of 1000 randomly generated task sets that could be scheduled.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



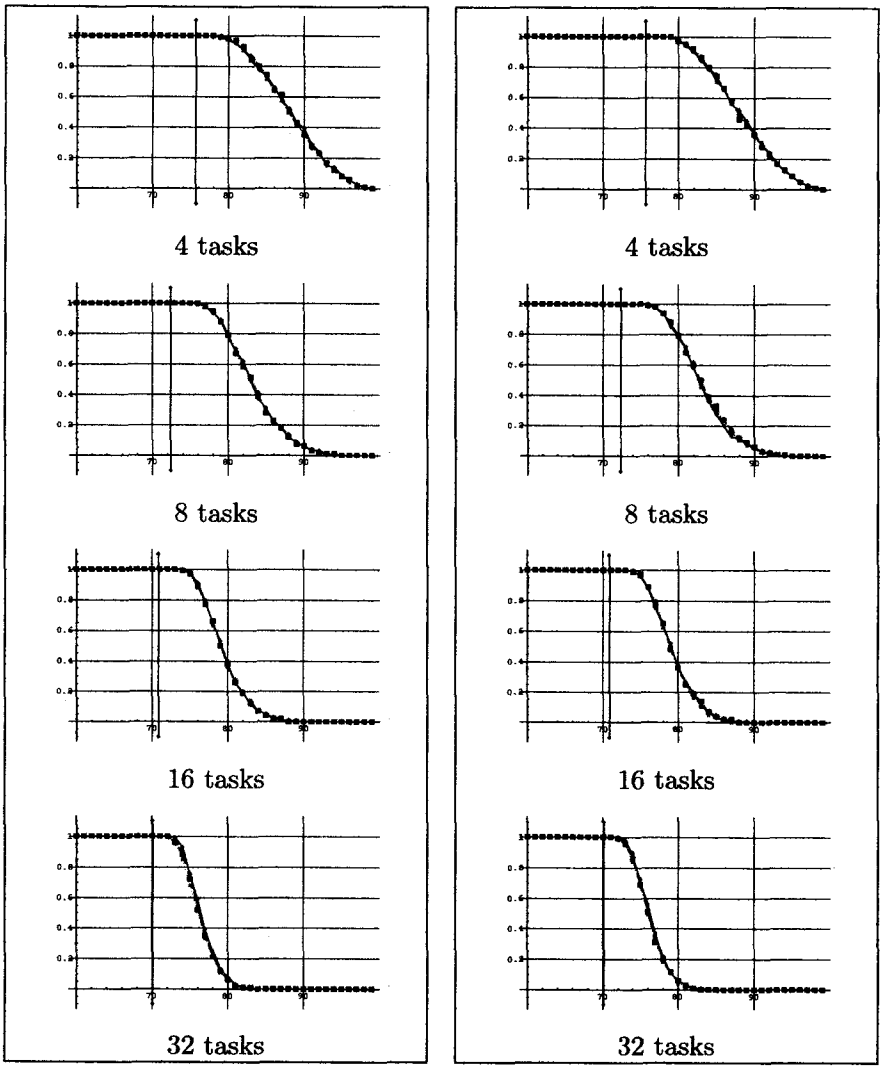

(a) Uniform distribution used for the random selection of the periods

(b) Indirect uniform distribution used for the random selection of the periods

Figure 3.9: RELIABILITY OF THE RMS SCHEDULABILITY TEST WITH NON-ZERO TASK SWITCH OVERHEAD: *The dependent variable of schedulability is a function of the total task set utilization (percentage values) and overhead size (single digit numbers in units of slot sizes). No correlation exists between the requests and periods of the tasks. The height at each point represents the percentage of 1000 tasks sets for which the critical region test was incorrect for the corresponding utilization-overhead combination.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



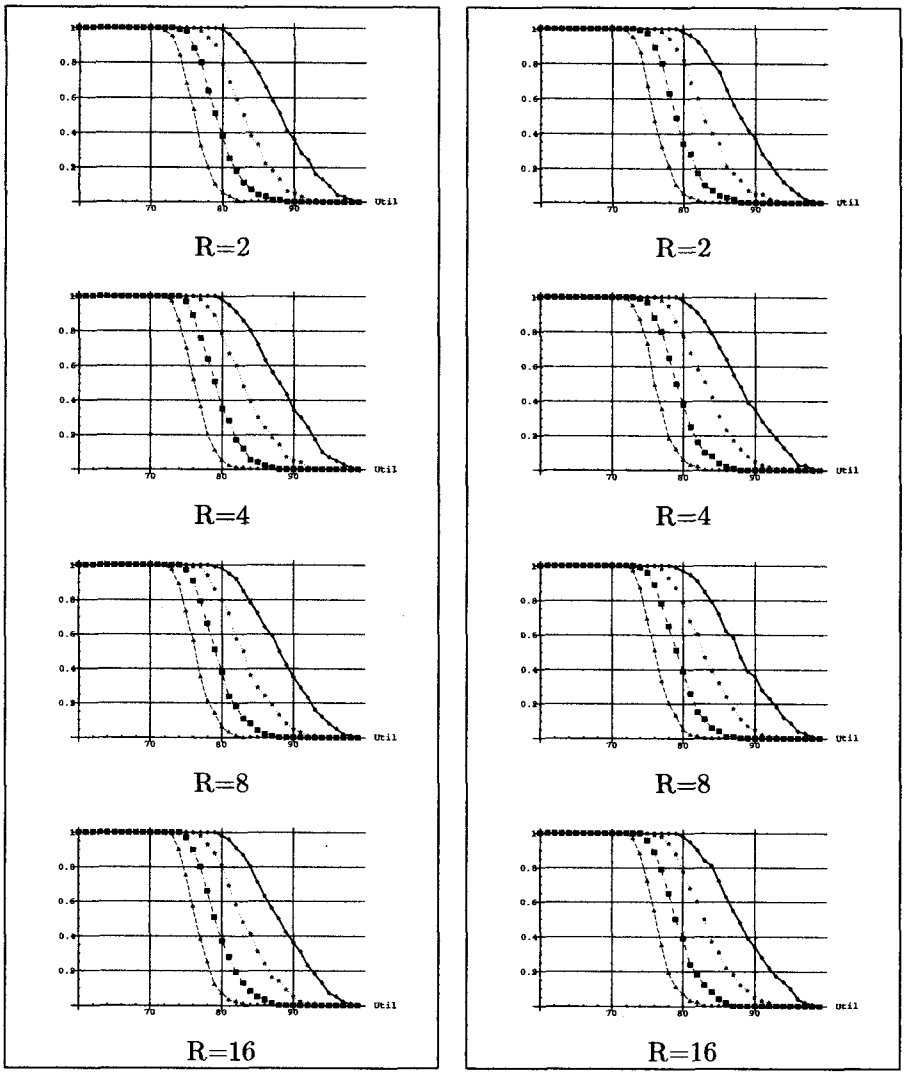

(a) Uniform distribution for the random selection of periods

(b) Indirect uniform distribution for the random selection of periods

Figure 3.10: DISTANCE CONSTRAINED SCHEDULABILITY PLOTTED AS A FUNCTION OF UTILIZATION (PM CURVES): *The dependent variable of schedulability is a function of the total task set utilization. No correlation exists between the requests and periods of the tasks. Separate curves in each plot represent* $\frac{\text{longest task period}}{\text{shortest task period}}$ *ratios of 2, 4, 8, 16, and 32. The vertical line to the left of all the curves (connecting two diamonds) shows the utilization based lower bound for the given number of tasks. Each data point represents the fraction of 1000 randomly generated task sets that could be scheduled.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



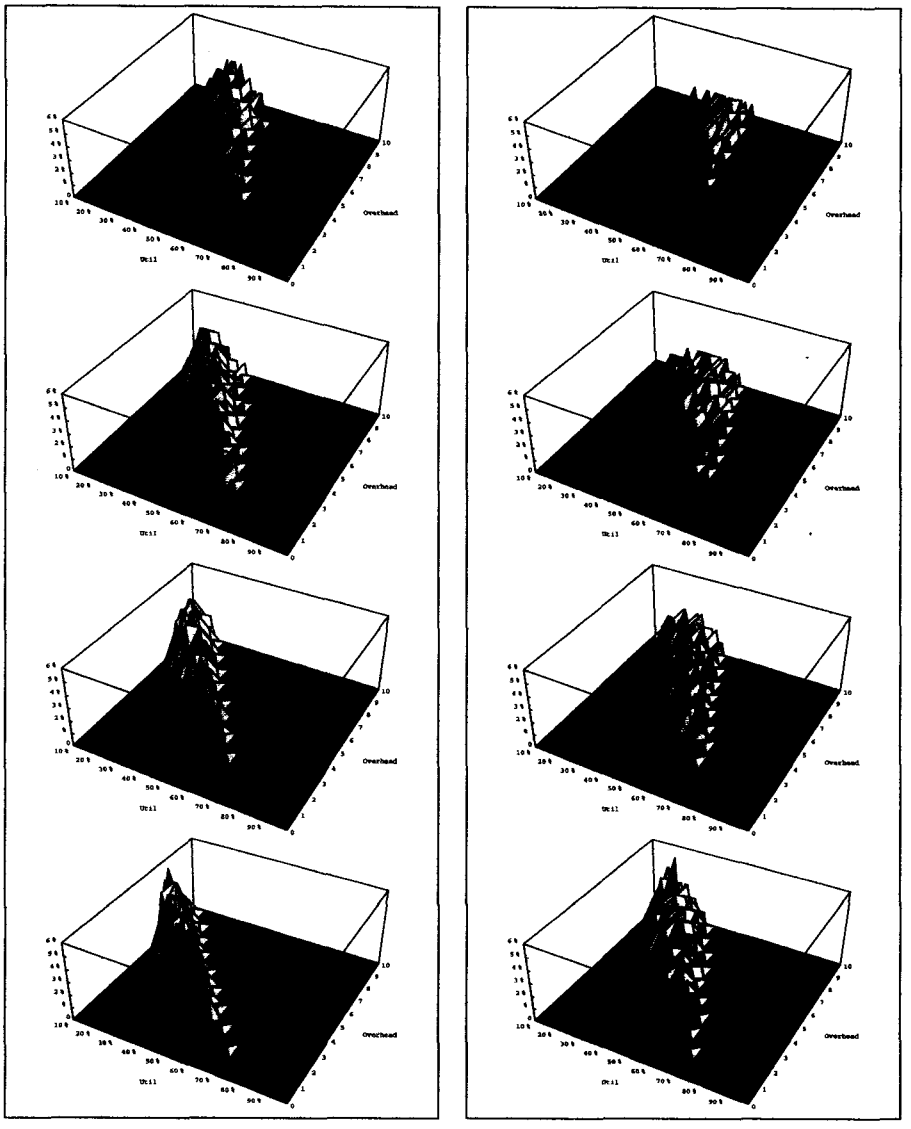

(a) Periods: Uniform distribution

(b) Periods: Indirect uniform distribution

Figure 3.11: DC SCHEDULABILITY PLOTTED AS A FUNCTION OF UTILIZATION (TSS CURVES): *These plots provide a different slice through the information shown in figure 3.10. Again, the dependent variable of schedulability is a function of the total task set utilization. No correlation exists between the requests and periods of the tasks. However, separate lines in each plot now represent different size task sets. Each data point represents the fraction of 1000 randomly generated task sets that could be scheduled.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



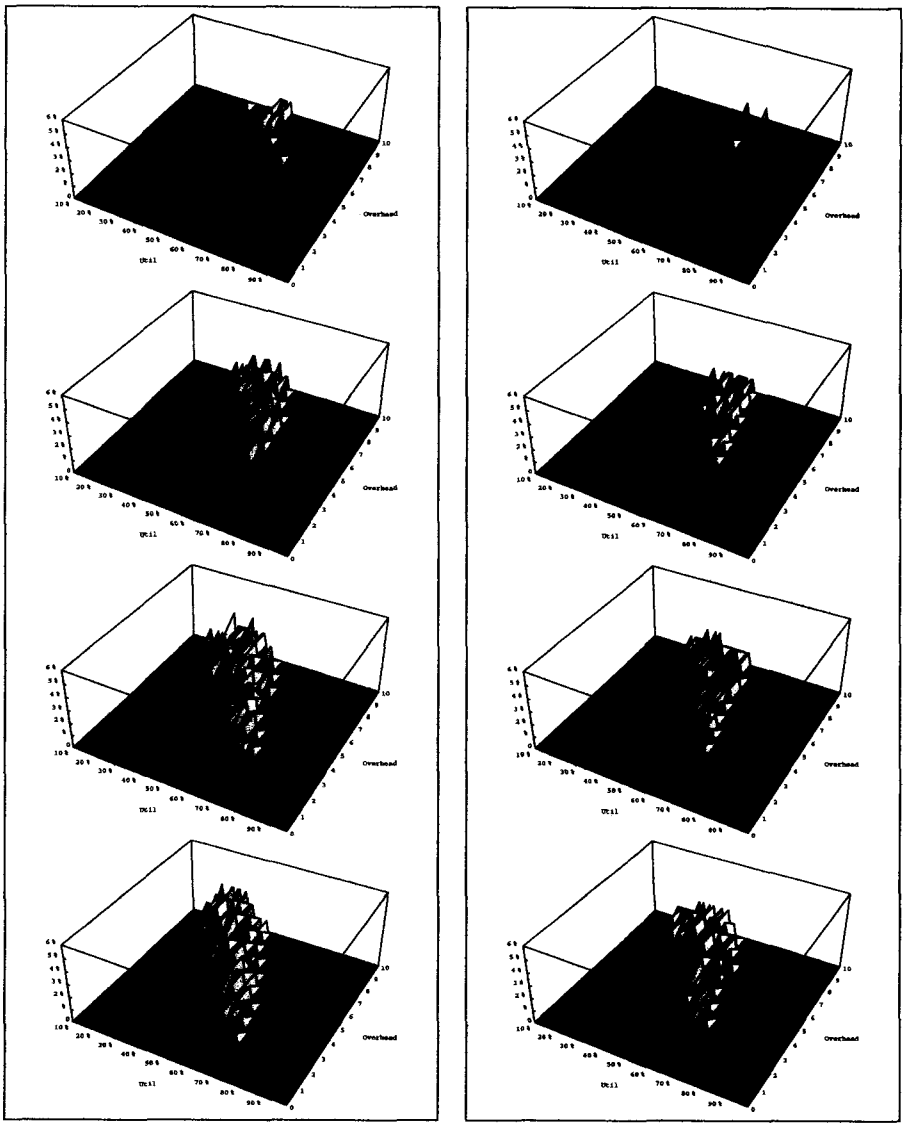
(a) $\frac{\text{longest task period}}{\text{shortest task period}} = 4$, task sets sizes (top to bottom) of 6, 8, 10, 12.

(b) $\frac{\text{longest task period}}{\text{shortest task period}} = 8$, task sets sizes (top to bottom) of 6, 8, 10, 12.

Figure 3.12: IMPACT OF SORTING ON DISTANCE CONSTRAINED SCHEDULABILITY, 1: *In each plot, the dependent variable of schedulability is a function of the total task set utilization (percentage values, range 10% to 90%) and overhead size (range 0 to 10). The range of the vertical axis is 0% to 6% and represents the schedulability difference (in percentage value) caused by sorting the tasks within specializations levels by their resource request; a positive height shows an improvement caused by the sorting. Each point is the average of 250 simulations.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



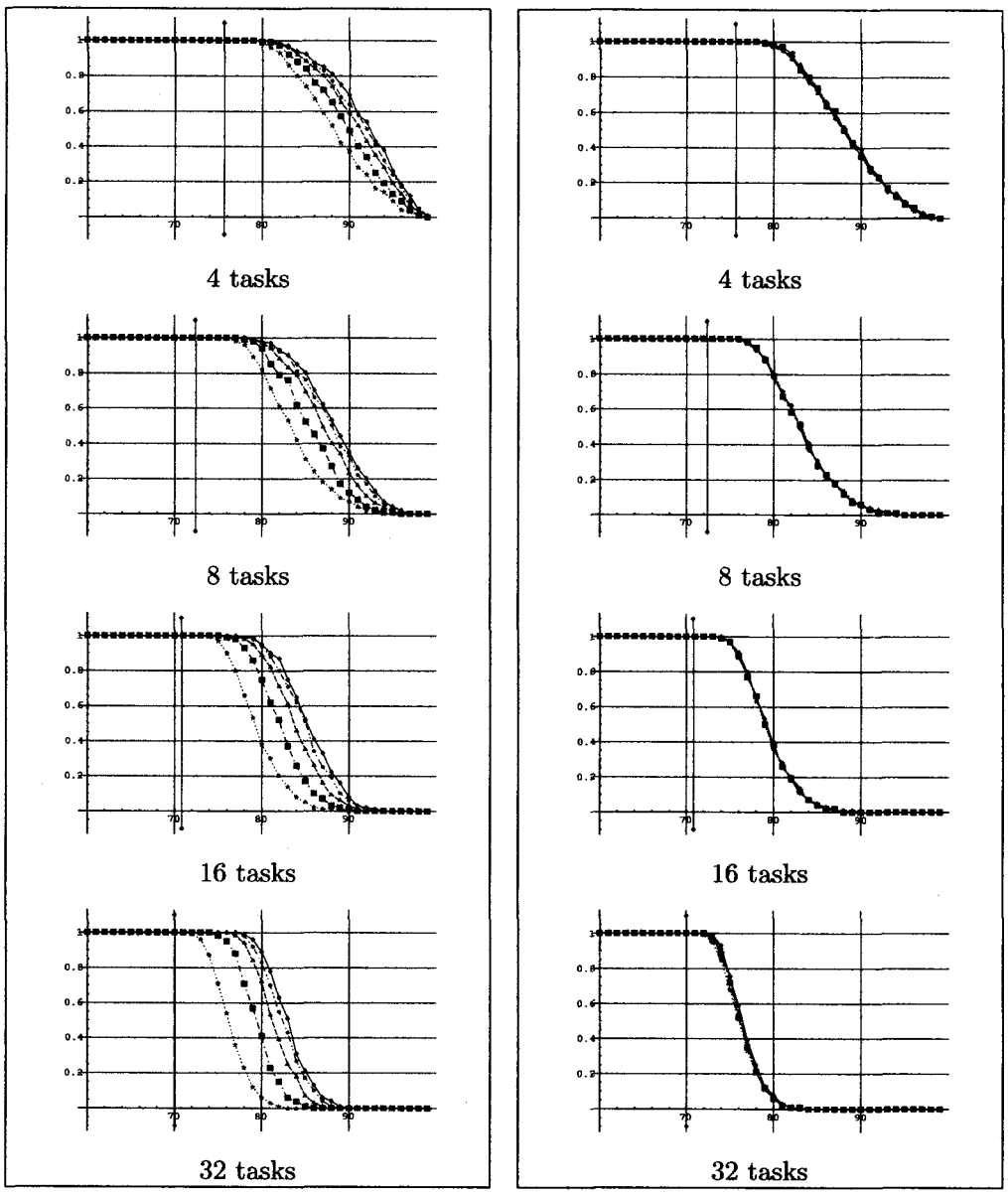
(a) $\frac{\text{longest task period}}{\text{shortest task period}} = 12$, task sets sizes (top to bottom) of 6, 8, 10, 12.

(b) $\frac{\text{longest task period}}{\text{shortest task period}} = 16$, task sets sizes (top to bottom) of 6, 8, 10, 12.

Figure 3.13: IMPACT OF SORTING ON DISTANCE CONSTRAINED SCHEDULABILITY, 2: *In each plot, the dependent variable of schedulability is a function of the total task set utilization (percentage values, range 10% to 90%) and overhead size (range 0 to 10). The range of the vertical axis is 0% to 6% and represents the schedulability difference (in percentage value) caused by sorting the tasks within specializations levels by their resource request; a positive height shows an improvement caused by the sorting. Each point is the average of 250 simulations.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

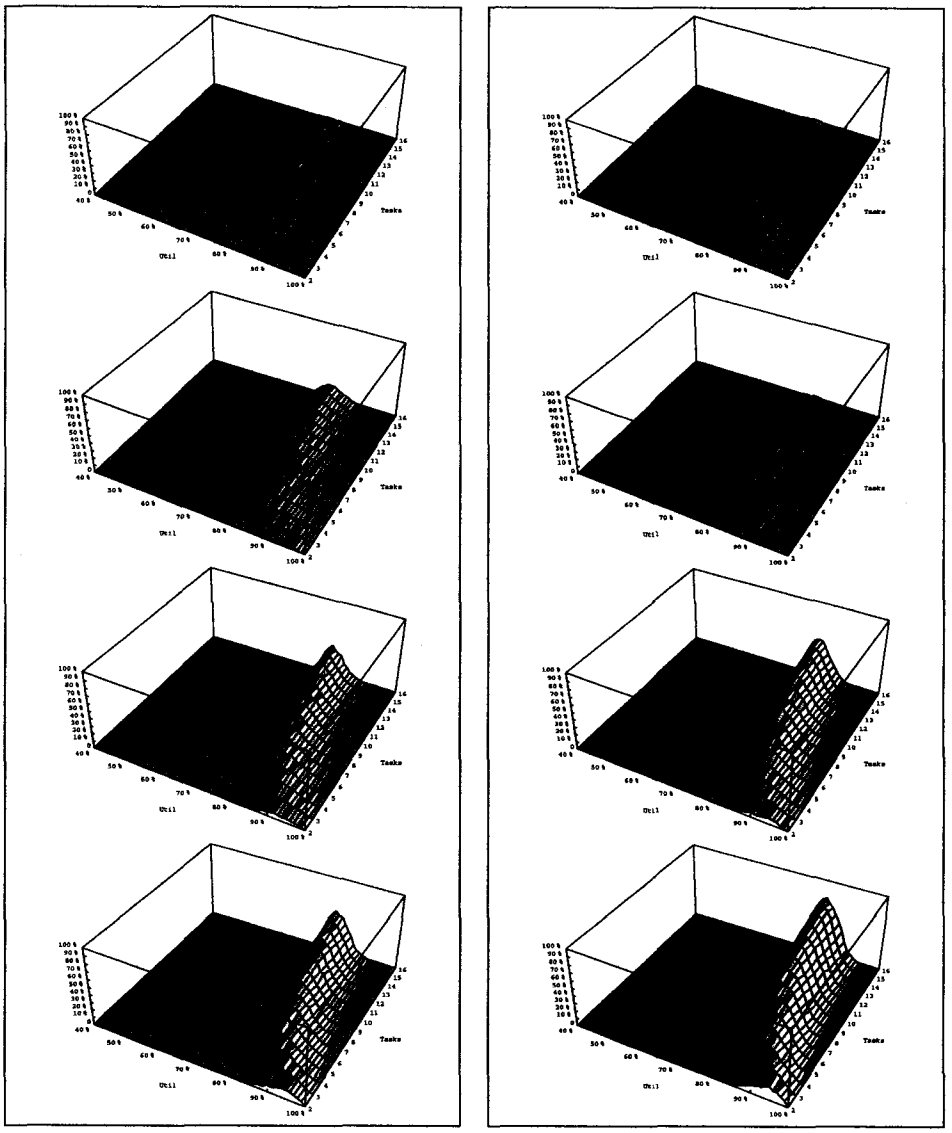



(a) Rate Monotonic Scheduling (b) Distance Constrained Scheduling

Figure 3.14: RATE MONOTONIC SCHEDULABILITY VS DISTANCE CONSTRAINED SCHEDULABILITY: *Figure 3.14(a) is a repeat of 3.7(a); figure 3.14(b) is a repeat of 3.10(a).*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



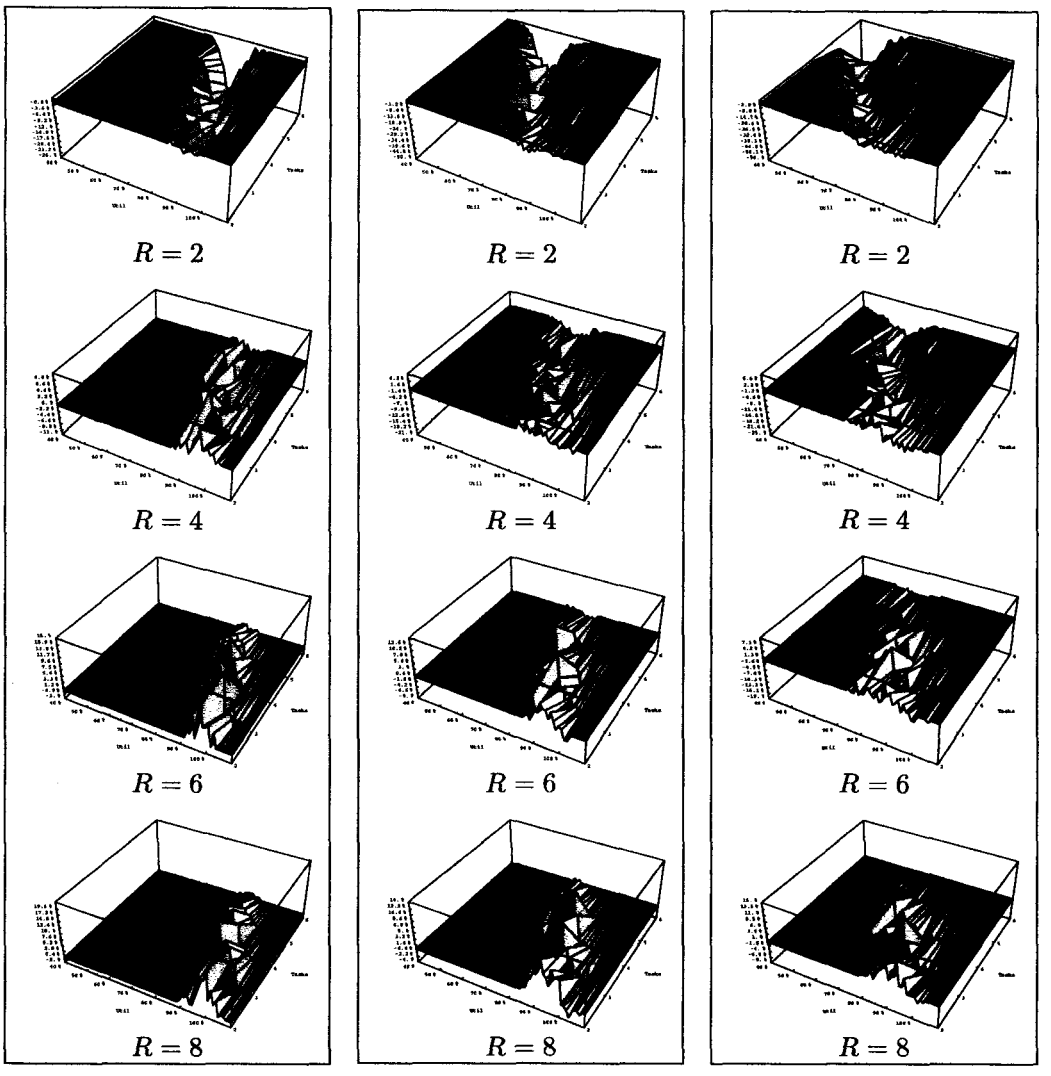
(a) Uniform distribution for the random selection of periods

(b) Indirect uniform distribution for the random selection of periods

Figure 3.15: RM SCHEDULING VS DC SCHEDULING WITH NO OVERHEAD: *The dependent variable is a function of the total task set utilization (40% to 100%) and the number of tasks (2 to 16). The range of the vertical axis is 0% to 100% and represents the schedulability difference between RMS and DCS based on an average of 1000 task sets; a positive value indicates that RMS outperforms DCS while a negative value (had there been one) indicates the converse. The task switch overhead is 0. The first, second, third, and fourth rows of plots represent* $\frac{\textit{longest task period}}{\textit{shortest task period}}$ *ratios of 2, 3, 4, and 8.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.




(a) Overhead = 1 (b) Overhead = 2 (c) Overhead = 3

Figure 3.16: RM SCHEDULING VS DC SCHEDULING WITH OVERHEAD: *The dependent variable is a function of the total task set utilization (40% to 100%) and the number of tasks (2 to 6). The range of the vertical axis differs but is a percentage representation of the schedulability difference between RMS and DCS based on an average of 1000 task sets; a positive value indicates that RMS outperforms DCS while a negative value (had there been one) indicates the converse. No correlation exists between the requests and periods of the tasks. The plots in the first column are from a task switch overhead of 1 while those in the second and the third columns are based on a task switch overhead value of 2 and 3. The first row of plots are based on a* $\frac{\textit{longest task period}}{\textit{shortest task period}}$ *ratio of 2, while the second, third, and fourth rows are based on 3, 4, and 8.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# Chapter 4

# Design of a Real-Time Data Link Layer Protocol

The goal of chapters 2 and 3 was to provide information on real-time scheduling algorithms which were originally developed for processor scheduling. We now focus on the development of a network protocol based on two of these algorithms. Because one of our goals is to provide deterministic media access, we start with a brief survey of related work (section 4.1). This is followed by a short description of the real-time nature of our own network protocol; since we assume no support from the underlying hosts and since real-time processing atop non real-time hardware could be dismissed as ill conceived, the few paragraphs provided in this short section are crucial in specifying the temporal goals of our work (section 4.2). After a description of the simulation system that we use as a proof-of-concept environment (section 4.3), we present the details of the network protocol and the operation of its scheduler (section 4.4).

Starting with this chapter, we will consistently use the phrase **periodic real-time stream** (or simply a "stream" when the full phrase becomes too monotonous) to refer to a network connection that delivers real-time data at regular intervals. In addition, each one of the periodic bursts will be called a **message** (we intentionally do not use the term **packet** for this purpose since they are many cases where messages are too large to be encapsulated



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

within the maximum transmission unit (MTU) of the underlying carrier protocol). As the name indicates, the two fundamental characteristics of a periodic real-time stream is that the constituent messages are transmitted in a cyclic manner and the delivery of each message is subject to a time constraint.

A periodic real-time stream in the context of networking is the analog of a task in real-time scheduling. The relation between these two concepts lead to additional terminology associations that we utilize throughout this document. The following four are the ones we most frequently use:

**Job** $\leftrightarrow$ Individual messages of a periodic real-time stream

**Task set** $\leftrightarrow$ Set of all periodic streams carried in a network segment

**Task set utilization** $\leftrightarrow$ Normalized total bandwidth required by the set of all periodic streams

**Task switch** $\leftrightarrow$ The process of switching from one periodic stream to another; we occasionally refer to this as a **source/stream switch** since "task switch" sounds alien within the context of networking

## 4.1 Deterministic Data Link Layer Protocols

This section is a brief survey of data link layer protocols that provide deterministic media access:

- **IEEE Standards: Token ring** (e.g. 802.5) and **FDDI** protocols make use of a circular topology and arbitrate message transmissions with the temporary possession of a token packet. Going with a compromise between standardized Ethernet (802.3) and token ring (800.5), the **Token Bus** protocol (802.4) arbitrates transmission rights with a token over a bus topology. The **Point Coordination Function** (PCF) of 802.11 is an optional capability that can be used to provide connection-oriented, contention-free services by enabling polled stations to transmit without contending for the channel.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- **Slot Based**: In real-time communication networks, performance guarantees for messages such as bandwidth allocations and transmission time bounds are provided by specialized hardware. Slot generators create fixed time intervals during which only a single node is allowed to send messages. A deterministic scheduler decides the allocation of slots to contending nodes. This makes it possible for each node to determine what percentage of the network bandwidth has been guaranteed. Most papers dealing with periodic real-time network scheduling and real-time communications assume this type of hardware support ([LS86], [BG91]).

- **RETHER**: In [VcC95], [cCV97b], [cCV97a], and [PC98], the authors propose, design, and apply a token passing scheme "Real-Time Ethernet" (RETHER) for bandwidth allocations over Ethernet. Their method is based on modifying the device drivers for the network interfaces to support two modes of operation: real-time (RT) and non real-time (non-RT). The non-RT mode is the same as the usual CSMA/CD protocol of Ethernet. The network switches to the RT mode when a node submits a bandwidth allocation request. A token is generated to cycle through two sets of nodes: the RT cycle containing the nodes that require bandwidth allocations and the non-RT cycle containing *all* the nodes of the network (the RT group is a subset of the non-RT group since the hosts with the processes requiring bandwidth allocations may also have non real-time processes). The token first cycles through the RT nodes. When each real-time node gets its share of the network, the token cycles the non-RT nodes. The system is configurable so that the latter cycle always gets a minimum percentage of the network use to prevent the starvation of non RT processes.

- Other: FieldBus ([HS95b]), Real-Time Ethernet Networking At Home ([FHH+02]).

## 4.2 Real Time Characterization of the System

In non real-time systems, software development can often be decoupled from hardware considerations. The primary reason for this convenience is that as long as the underlying



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

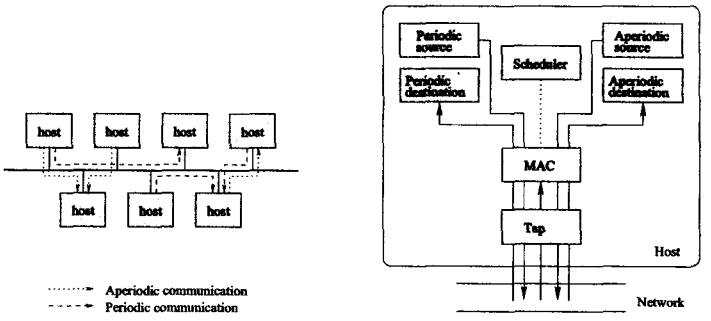
hardware supports a number of fundamental operations, software can often make up for what's missing (the Java virtual machine is a good example for this). In real-time systems, the situation is different. If a software system needs to produce $X$ but finds no support from the underlying hardware for the timely production of the partial results that lead to $X$, then temporal correctness is unattainable.

At the device level, temporal constraints can sometimes be accommodated on non real-time hardware. For example, in order to guarantee that a transmission does not exceed a particular duration, an equivalent upper limit in units of bits can be enforced by any trivial buffer management scheme (with a transfer rate of $R$, an upper limit of $t$ units of time is equivalent to $tR$ bits of transmission). At the system level, however, devices are under the control of the operating system, preventing user level entities to establish time bounds between the submission of requests and their completion. This brings up an important question regarding our network protocol: if the hosts on which we operate are not real-time systems, then how can we make any guarantees for the timely delivery of the messages of the periodic real-time streams?

One option is to in fact assume the availability of a real-time operating system. Indeed, during the last 10 years, a number of UNIX flavored real-time operating systems have been developed, making this assumption a fairly reasonable one. Yet, in terms of the intellectual appeal of the work involved, this solution is less than satisfying since it simply relegates the problem to someone else. The alternative approach taken by our protocol is based on the creation of schedules that are specific to the responsiveness of the hosts at the instant the schedule is created. Subsequently, as the schedule is used, only the hosts that remain at the same level of responsiveness get the bandwidth they require (and thus the deadline guarantees); those who slow down see a reduction in their allowance (if the scheduler itself falls behind, we employ a "schedule migration" function which will be discussed later in this chapter). This setup is clearly more lax than the ones we find in scheduling literature that typically manage nuclear reactors, airline controllers, etc. However, it does offer a compromise service between the best-effort and the conventional-hard-real-time extremes.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.




(a) Segment view (b) Host Structure

Figure 4.1: SIMULATION ENVIRONMENT: *The dashed lines in part 4.1(a) represent periodic real-time streams between pairs of hosts in a virtual segment. Since we have a shared communication medium, the point-to-point representations of the transmissions is an illusion at the application layer. The structure of each host is outlined in part 4.1(b). The generation and destruction of packets (periodic and aperiodic) are managed by four modules. New schedules are constructed by the* SCHEDULER *module when a request for a new periodic real-time stream is received. The created schedule is then used by the* MAC *module which coordinates the transmission sequence within the network. The* TAP *module mimics some of the physical layer aspects of the network interface as well as intentionally slowing down the operation of the protocol when the network is underloaded (details within the text).*

In terms of quantifying the responsiveness of hosts, the protocol makes use of interrupt processing times (which is the interval of time from the instant an interrupt is raised to the completion of its service). These measurements have limited value in individual and absolute terms. But the protocol relies more heavily on the observed changes and it uses an auto-regression mechanism to be immune to intermittent fluctuations. This allows the protocol to reduce various forms of difference (i.e. raw hardware speed, current computational load, etc) to a single scalar which is then used during the construction as well as the use of the schedules (details in section 4.4).

## 4.3 Simulation Environment

In this section, we provide an overview of the virtual environment in which our simulations were conducted. This is necessary for two reasons. First, the provided information creates



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

context for the presentation of the network protocol in section 4.4. Second, we need to show that hosts within the virtual environment are sufficiently realistic approximations of actual hosts; otherwise, we cannot claim any generality for the results.

Each simulation (designed in OPNET) runs on a virtual segment of an arbitrary number of hosts. As shown in figure 4.1(a), a **segment** represents an interconnection based on a shared wired/wireless medium without any repeaters, bridges, or routers; Aloha, CSMA, CSMA/CD, CSMA/CA, etc, all fit this description. The key element within a segment is the host model. Each host is assumed to be equipped with basic networking hardware that allows that host to send and receive messages over the shared network in CSMA mode. The ability to operate in this minimalistic and stateless manner provides the basis for bootstrapping the real-time protocol as well as restarting it if a temporary interval of frequent message loss leads to cascading deadline failures. At any point in time, one of the hosts functions as the **scheduler** for the entire segment. It can operate as a **distance constraint scheduler** or a **rate monotonic scheduler**. In addition to periodic real-time streams, the hosts are also capable of generating messages representing aperiodic traffic.

As shown in figure 4.1(b), the host model consists of seven modules which collectively perform three types of functions:

- **Generation of periodic traffic**: The PERIODIC SOURCE module generates the periodic real-time stream traffic whose timely delivery is assumed to be the most important function of the underlying network. Any well known distribution can be used to specify the size and inter-arrival times of these messages. Each periodic source module can initiate and terminate an arbitrary number of periodic real-time streams, though multiple streams from the same module cannot be simultaneously active (i.e. stream $i$ of a host must be terminated before stream $i + 1$ of the same host can start). The PERIODIC DESTINATION module receives the real-time messages sent to the underlying host and collects statistics to quantify the timeliness of their deliveries.

- **Generation of aperiodic traffic**: Two types of aperiodic traffic are generated.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The first type is a product of the operation of the real-time protocol itself. Since the protocol processes are distributed over the network, control messages must be exchanged between the hosts. For example, when a host initiates a new periodic real-time stream, the SOURCE module of that host sends an aperiodic message to the scheduler host to transmit the associated admission request. The second type is for simulating the background traffic we would find in every real network (routing updates, name lookups, etc). These messages are generated by the APERIODIC SOURCE module and consumed by the APERIODIC DESTINATION module. As in the PERIODIC SOURCE case, the rate and message size characteristics of the APERIODIC SOURCE can be customized with host level configurations.

- **Construction and use of the schedule**: Each host has the capability to function as the scheduler for the segment. The SCHEDULER module creates new schedules as it receives admission requests for new streams. The MAC module makes use of the created schedule to determine the message transmission sequence for all the hosts.

  These two modules also represent the overall protocol hierarchy split within the design. The higher level functions performed by the SCHEDULER are assumed to take place within the kernel space of a host while the MAC module represents lower level logic that can be embedded in a network interface (e.g. the automatic and continuous reading of the protocol fields piggy-backed on specific user data messages).

The seventh and final module called the TAP is used to slow down the operation of the protocol when the network is underloaded. Although this sounds counter-productive, it makes up for a "braking" mechanism that is usually missing in periodic scheduling algorithms. As will be detailed in section 4.4, preventing the source switching rate from exceeding the minimum required to meet all deadlines reduces the computational cost of the protocol.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

## 4.4 Network Protocol

In this section, we outline the operation of a network protocol that provides the type of real-time service defined in section 4.2. We limit the information to algorithm-neutral issues; algorithm specific discussions as well as the simulation results are reserved for chapters 5 and 6.

The real-time protocol supports two modes of operation. In **non real-time mode**, media access is coordinated by a rudimentary CSMA mechanism. Despite its limitations, the non real-time mode provides the hosts a basic communication facility that is sufficient to bootstrap the network and maintain low intensity aperiodic traffic. In **real-time mode**, media access is coordinated by a deterministic scheme. The host that initiates the first periodic real-time stream creates a schedule that determines the order, type, and duration of every transmission within the segment. From that point on, each new stream leads to the creation of a new schedule (though it is not the same scheduler that creates them all; hosts take turns in doing this). When all the network connections are terminated, the protocol returns back to the non real-time mode.

In this high-level view, our system looks very similar RETHER [VcC95] and thus we emphasize a few of the differences. Prior to the real-time mode, RETHER operates by CSMA/CA while ours falls back to the much simpler CSMA. Within the real-time mode, however, RETHER operates in a strictly cyclic mode while our system makes use of Distance Constrained Scheduling (DCS) or Rate Monotonic Scheduling (RMS). The focus of RETHER is the incorporation of deterministic coordination within an actual Ethernet implementation albeit in the simplest way possible. The focus of our project (even though it is confined within simulations as opposed to being a real implementation) is the exploration of the applicability of real-time scheduling algorithms. As the next two chapters will show, we look into the consequences of probabilistic variations in request size, message generation time, and stream start time. We also explore ways to make the real-time schedules more resilient to these variations.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Since the primary purpose of this project was to explore the applicability of real-time scheduling algorithms on real-time network traffic, most of our results are obtained within the real-time mode. In fact, as far as the results of chapters 5 and 6 are concerned, the sole purpose of the non real-time mode was to start the real-time mode in a systematic manner after the virtual hosts were first activated.

### 4.4.1 Schedule Representation

In real-time systems, the completion of a scheduling event requires the scheduler to consider the state of the task set and determine which task should get the shared resource next. This state information is always assumed to be readily available whether the underlying system is a processor (RMS [LL73]), a satellite transmission system (pinwheel scheduling [HMR+89]), ATM switch (statistical RMS [AB98]), or a tightly coupled multiprocessor (distance constrained scheduled [HLH96a]). In the case of network scheduling, however, it is not; the source of each periodic real-time stream is an independent process on a different host. Since a conventional network of heterogeneous hosts does not provide shared memory or any similar facility to capture a global snapshot, the scheduler cannot conveniently determine which subset of the periodic real-time streams are ready to transmit.

The distributed nature of the system, the amount and time-granularity of the required information per scheduling decision, and the lack of hardware support of any form all make it apparent that there is no easy solution to this problem. Without an effective solution, the theoretical benefits of many scheduling algorithms are hard to attain. For this reason, our protocol follows the alternate route of *generating* a close enough version of this information at the scheduler rather than fetching it from the network. When a new stream is admitted, the scheduler creates a structure to represent a major cycle of the current periodic real-time stream set. At each instant, this structure determines the type, source, and duration of the current transmission is a trivial lookup instead of an expensive computation. Compared to *computing* the details of the next event, this approach is clearly inefficient in terms of space but it does have a number of advantages:



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1. The schedulability tests that accompany scheduling algorithms occasionally lead to incorrect conclusions when the task switch overhead is greater than zero (details in chapter 3). The construction of the schedule allows the scheduler to avoid these problematic situations.

2. The construction of the schedules also allow them to be fine-tuned. We will see an example of this in chapter 6 for DCS where a minor variation in the basic operation of the algorithm will lead to better management of variations in message sizes.

3. Unless a schedule is constructed in advance, the scheduler has no way of knowing the time until the resource gets reassigned to a different task. This makes it possible for tasks to get the resource for extremely short intervals of time. In theoretical models (or tightly coupled systems), this is not a significant problem. But when the task switch adds up to a substantial overhead, then total utilization of the resource can significantly drop by short events. The creation of the schedule before its use allows the scheduler to avoid these situations and, in the case of networking, prevent the transmission of messages containing just a few bits of data.

4. Perhaps most significantly, the cost of using the schedule after its construction is O(1) per event while algorithms that compute the next event (with the luxurious assumption of having the information they need) scales with the number of tasks in the schedule (linear complexity being a trivial lower bound).

Since the schedule data structure is a local entity of the scheduler host, a token packet is used to communicate the scheduling decisions to the remaining hosts of the segment (our first scheduler was based on replicating the schedule throughout the segment allowing each host to determine its current-event specific downstream neighbor with a *local* lookup [ER03]; this approach worked but also needed a complicated messaging scheme to keep the separate copies of the schedule synchronized). When it is time for a host to send data, the scheduler sends a token packet to that source and specifies the type and maximum duration of the



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

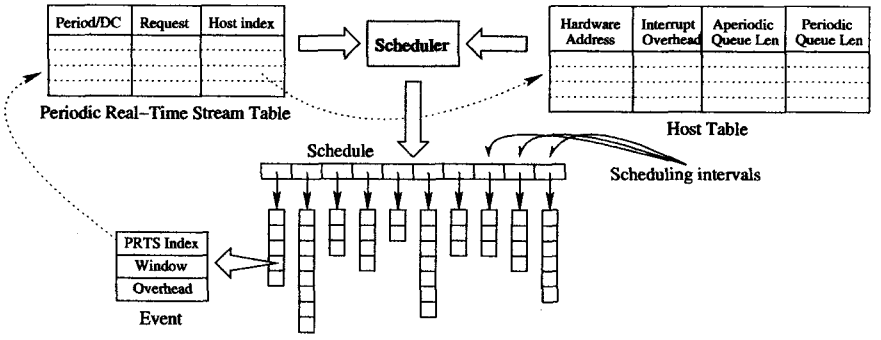



Figure 4.2: CONSTRUCTION OF A NEW SCHEDULE: *The construction of a new schedule is based on information from two tables maintained by the network protocol. The parameters of the ith periodic real-time stream is stored in the ith row of the table on the top left. The 'Host Index' field of this row is used to cross-reference the associated host from the table on the right. Assuming that the current periodic real-time stream set is schedulable, the resulting schedule is arranged in terms of* ***scheduling intervals****, each of which is made up of one or more* ***events****. Each event corresponds to a particular message (or fragment of a message) of a particular stream. The 'PRTS Index' field is a cross-reference to the associated periodic real-time stream while the 'window' and 'Overhead' fields represent the upper limit for a transmission and the total cost of letting that stream take its turn. Due to the two cross reference fields (shown with the dashed lines), the asymptotic cost of the lookups performed by the scheduler per event is* $O(1)$ *regardless of the number of streams currently carried.*

allowed transmission. During the real-time mode, hosts are allowed to make transmissions only when they receive this token.

### 4.4.2 Schedule Creation

As illustrated in the bottom half of figure 4.2, a schedule is made up of a sequence of **scheduling intervals**, each consisting of one or more **events**. An event represents the time interval during which a particular stream is allowed to make transmissions. For the stream with the highest priority, an event contains an entire message. For lower priority streams, events usually represent a fragment of a message due to the preemptive operation of the scheduler. Although the boundaries of scheduling intervals depend on the particular scheduling algorithm used, they are in all cases used as synchronization mechanisms between the schedule and the network. For example, in the rate monotonic version of the protocol,



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the scheduler allows the hosts associated with the starting events of scheduling intervals to wait (without losing their turn) until a message is generated. Scheduling intervals are also used as boundaries in distributing the lost/saved time at the completion of an event (more information on this will be provided later).

The creation of a new schedule depends on two data structures maintained by the protocol. The **periodic real-time stream table** shown in the top left portion of figure 4.2 maintains the source, period, and request fields of each currently active stream. The **host table** shown in the top right portion of figure 4.2 maintains various parameters of each host in the segment. During the construction of the schedule, the overhead associated with each event partially depends on the interrupt overhead field of the associated host as well as the transmission rate of the underlying network.

When the current scheduler receives a request for the admission of a new periodic real-time stream, it checks recently gathered state information about the hosts and picks one whose computational load appears to be low. It then sends the current set of periodic real-time streams (including the requested one) as a request for the generation of a new schedule. If the chosen host can create a new schedule, it inherits the scheduling responsibility. Otherwise, the request is rejected and the current scheduler continues its operation. This scheme increases the complexity of the network protocol but it also provides four substantial benefits:

1. It splits the *creation* of the schedule from its *use*, offloading the scheduler from a fairly taxing process. We cannot unfortunately quantify the time gained since our work is tested in a simulation rather than an actual system. However, it is reasonable to assume that any reduction in the computational load of the scheduler would be a boost to its stability and reduce the variations in its response during its scheduling operations.

2. It leads to the distribution of the protocol overhead (albeit at a crude level) since no single host remains to be the scheduler for extended periods of time.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3. It allows the scheduler to pro-actively initiate the transfer of the scheduling responsibility to another host when its own responsiveness falls below a threshold (this could happen, for example, if a new computationally taxing process starts). This provides a partial solution to the problem of the scheduler itself becoming a reason why deadlines are missed. We say a "partial solution" as opposed to a "full solution" since, as usual, it is not possible to provide total temporal guarantees as long as the underlying hardware does not provide true real-time services.

4. It prevents the system from having a single point of failure. Although this feature is not fully implemented, the existing mechanisms can be extended so that a *subset* of the hosts contain the descriptions of the current periodic real-time stream set; if the current scheduler fails to generate a token by an established threshold, then one of these hosts could either assume the scheduling responsibility or select the next scheduler by sending the task set to the selected host. In the simplest case, this subset would contain only the host that used to be scheduler for the schedule prior to the current one. Since this host already knows the current stream set (it was the one which received the new stream request) and has a fairly recent host state table, it automatically assume responsibility without the need to communicate any control information with other hosts. But if this host is down as well, then the network can abort the real-time mode and fall back to CSMA to start from scratch.

The protocol's ability in coordinating a heterogeneous network of hosts hinges on maintaining host specific responsiveness values which lead to event specific overhead figures. However, without any limits, it is possible for a host with an extremely high interrupt processing overhead to collapse the entire system by rendering every future stream set unschedulable. For this reason, the scheduler rejects stream admission requests coming from hosts whose interrupt processing overhead exceed a configuration threshold. As we will see later in this section, this threshold is also useful in scheduling the aperiodic reserve stream that manages the aperiodic traffic of the system.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

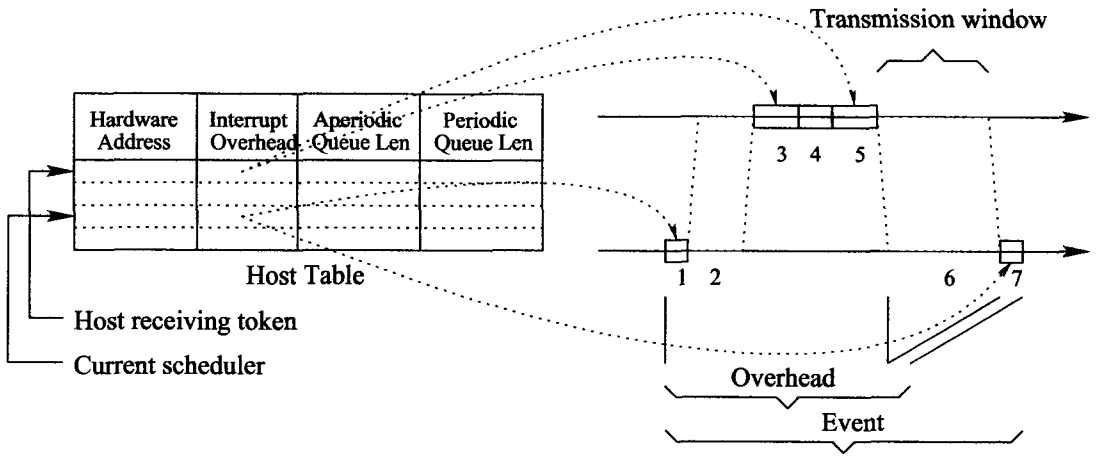



Figure 4.3: "EVENT" DEFINITION: *The left part of the diagram displays the host table where the scheduler maintains identification (hardware address) and performance (an auto-regression based measure of the hosts interrupt processing time, aperiodic and periodic queue lengths) attributes for each host within the segment. The right part of the diagram shows the event definition in detail. The temporal length of the event is broken into seven stages: (1) token send interrupt at the scheduler, (2) token transmission and propagation through the network, (3) token reception interrupt at the host, (4) data packet preparation at the host, (5) same as 3, (6) data packet transmission and propagation through the network, (7) same as 1. It is only the sixth stage that represents actual work; the remaining ones are part of the "task switch overhead" as the scheduler switches from one periodic real-time stream to another.*

Figure 4.3 shows the definition of an event and the computation of the associated overhead. Of the seven stages that take place from the generation of one token to another, the first, third, fifth, and seventh ones are read directly off of the host table. The second and the sixth ones depend on the size of the token and the resulting data message(s) sent by the host receiving the token. Finally, stage 4 represents computational time for low level functions such as the preparation of the messages (and the corresponding packets), buffer managements, etc.

### 4.4.3 Schedule Use

As stated earlier, the scheduler cycles through the schedule data structure and determines the source, type, and duration of the current transmission. Through the cross-references



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

between the constituent data structures, the asymptotic time complexity of the scheduler per event is kept at $O(1)$ regardless of the current number of streams. The schedule remains in use until a request for a new periodic real-time stream is received.

Since each event represents a message or a fragment of one, it initially seems as though the reception of a token leads to one transmission at most. In general, however, multiple packets are sent per token reception for two reasons. First, the message-centric construction of the schedule does not take into account the maximum transmission unit (MTU) of the underlying network and thus usually creates messages that exceed the MTU limitations. Second, hosts occasionally experience a single message build-up which can only be transmitted with excess capacity. If any slack exists (e.g. if the current message is smaller than expected) and if an earlier message has been queued up, then the host has the freedom to make multiple transmissions as long as the upper limit found in the token is not violated. Regardless of the number of packets sent, the MAC layer of the sender marks the last packet by setting its '`last`' to '`TRUE`'. This helps the scheduler in determining the end of the current event.

The discussion of the protocol has so far indicated that schedules are created when requests for new periodic real-time streams are submitted. Naturally, when these requests do not arrive often, the average lifetime of schedules increase. And if the message sizes of streams, ready times of the messages, and computational loads of the hosts change with time (as we would expect in any real network), it is unrealistic to assume that a static sequence of scheduling decisions will effectively control the network for any non-trivial interval of time. For this reason, two mechanisms are built into the protocol to update the schedule in reaction to some of these changes, short of recreating a new schedule. Here, we present the details of the first one which is based on updating event definitions with the most recent host responsiveness values; in chapter 5 and 6, we explain the second mechanism as part of the idle time management part of the algorithm specific components of the protocol.

As we saw earlier, the scheduler determines the token passing overhead of a host by making use of its responsiveness. The estimation of this value starts when a MAC layer



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

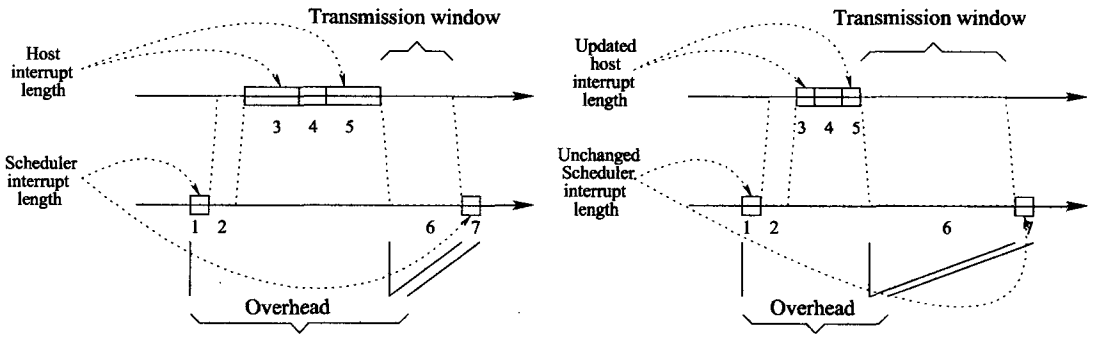



Figure 4.4: UPDATED OVERHEAD: *The left part of this figure shows the definition of an event with the host responsiveness values available during the construction of the schedule. During the use of the schedule, however, we assume (for the sake of this example) that the interrupt processing overhead of the associated host drops. As shown in the right part of the figure, this leads to a recomputation of various values under the main constraint that the overall duration of the event remains a constant. In this particular case, the changes lead to a longer transmission window which offer a greater portion of the bandwidth to the associated host "for its good behavior". This could of course go the other way and hosts may see a reduction in their transmission windows due to reduced responsiveness. The protocol allows transmissions as short as a single bit of user data. One may wander the value of such little net transfers... The reason they are allowed is that the protocol still receives valuable host state data piggybacked on packets that are otherwise on the verge of being useless as far as data transfer goals are concerned.*

entity measures the duration of the interrupt processing time during the reception of a token. This measurement is used as the most recent value fed into an auto-regression function

$$\text{ioh}(t) = \alpha s(t) + (1 - \alpha)\text{ioh}(t - 1)$$

which produces the responsiveness value stored by the scheduler (second column in the host table of figure 4.2). These values are then communicated to the scheduler through a protocol field piggy-backed on the packets whose '`last`' field are set. The updating of a scheduling event based on the new host responsiveness measures are shown in figure 4.4. As a consequence of this scheme, hosts that run slower than the rate at which the schedule was created are not allowed to cause deadline failures at other hosts.

The work required by the scheduler to receive the host responsiveness values is clearly network intensive. However, we do not expect it to be an overwhelming process for a



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

number of reasons. First, the scheduler needs to read the responsiveness values only off of a subset of the data packets (i.e. the ones which have the '`last`' field is set). Second, the processing of these values are trivial; it amounts to a single update in a table. Therefore, it is reasonable to assume that this functionality can be performed in the firmware of the network interface card without causing much disruption for the processor. Finally, if the scheduler is still overwhelmed, it may choose to consider only a subset of the packets carrying this information. Although this is not the ideal case for maintaining the most up to date information about the hosts, it offers a trade off between scheduling overhead and accuracy so that any host may operate as the scheduler for its segment.

### 4.4.4 Handling of Aperiodic Traffic

Although the bulk of the messages during the real-time mode of the protocol is periodic, some percentage of the traffic unavoidably remains to be non-periodic (or **aperiodic** as is commonly referred to in the literature). This is primarily caused by fundamental networking functions such as name lookups and routing updates that we assume to be needed by any real network. But a second contributing factor is the operation of the real-time mode itself. For example, the formation of a new periodic stream at a host can be communicated to the scheduler only through an aperiodic message. For these reasons, the network protocol we propose provides a best-effort mechanism to deliver aperiodic messages.

The entity that allocates bandwidth for the aperiodic messages is a special periodic stream called **aperiodic reserve**. The aperiodic reserve is present in every schedule and is characterized with the same parameter set that defines an ordinary periodic stream. For example, if the scheduler is based on the rate-monotonic algorithm, then the aperiodic reserve is associated with a period and a resource request. Its scheduling and transmissions, however, require special treatment:

- SCHEDULING OF THE APERIODIC RESERVE EVENTS: As was shown in figure 4.2, the host associated with a periodic stream is known at the time the schedule is created. The aperiodic reserve, however, is just a *place-holder* for an aperiodic transmission;



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

each of its events are associated with a particular host only *at the instant* that event is encountered by the scheduler. In dealing with this "under-specified" stream, the protocol uses a different scheduling logic. First, the host responsiveness value of the events is set at the maximum limit (as will be shown further down, this does *not* lead to an under-utilization of the network). This way, none of the hosts that get associated with specific aperiodic reserve events can "surprise" the scheduler with a responsiveness value that is larger than expected. Second, the aperiodic reserve is assumed to be successfully scheduled if it gets a specific fraction of the requests it makes; this is a simulation/protocol configuration value and has been set at 50% during the running of the simulations, though any value between 1% and 99% is acceptable.

- TRANSMISSION OF THE APERIODIC RESERVE MESSAGES: The association of each aperiodic reserve event with a particular host is done though an auxiliary data structure. By using different auxiliary data structures and by picking different parameters (periodic, resource request, etc), the user has control over the granularity and responsiveness of the system for aperiodic transmissions.

  The simplest auxiliary data structure for the aperiodic reserve is a cyclic list creating a round robin order between the hosts; if there are $N$ hosts in the network, then each host is guaranteed to make one aperiodic transmission for every $N$ aperiodic reserve events of the schedule. However, a more flexible scheme is based on the use of a priority queue; figure 4.5 shows the details of this arrangement. Even though the complexity of the aperiodic traffic processing is raised from $O(1)$ to $O(\log_2 N)$ per aperiodic transmission, the establishment of a priority scheme allows, for example, the real-time protocol related aperiodic messages to be favored over those created by ordinary application layer processes (it is important to note here that the increase from $O(1)$ to $O(\log_N)$ is effective *only* for the aperiodic message processing; regardless of the auxiliary data structure used, the processing of the *periodic* messages remains at $O(1)$).



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The priority function in the current version of the protocol uses the periodic queue lengths, aperiodic queue lengths, and computational loads of the hosts. However, this is more for providing a proof-of-concept argument than a rigorous study of the best priority function.

As was the case with host responsiveness measures, these values are read off of the data packets whose '`last`' fields are set. In order to prevent aperiodic transmission request starvation, the scheduler raises the effective value of the last-aperiodic-transmission parameter with time. This solves a secondary yet related problem: if scheduler learns the other hosts' computational and communication load based on their most recent transmissions, how will it know about the state of the hosts that are currently quiet? Since the host ids of *all* the hosts (rather than just the ones which are known to have any traffic accumulation) are placed in the priority queue and since the last transmission time gains weight with time, every host is eventually granted the chance to make an aperiodic transmission. Even if a host has no data to transmit, receiving an empty packet from it useful for the scheduler. As mentioned earlier, the scheduling responsibility migrates each time a new schedule is generated as a result of the addition of a new traffic source. The scheduler's ability to pick a lightly loaded host for the migration increases if it has up to date knowledge of the CPU load of all the hosts in the segment.

The underutilization of the network due to the use of the worst case host responsiveness values for the aperiodic reserve events is prevented during the *use* of the schedule. As soon as the host associated with an aperiodic reserve event is determined, the scheduler uses the event definition update technique outlined in figure 4.3. That is, the responsiveness value of the host replaces the worst case value used during the scheduling which almost always leads to aperiodic transmission longer than planned.

Aside from being a necessity, the aperiodic reserve is also useful for the operation of the



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

network protocol for two reasons. First, it can be used to enforce an upper limit on the message sizes. We had earlier mentioned that the scheduler is oblivious to the MTU of the underlying network; if the shortest period in the stream set as well as its associated requests are sufficiently long, then the scheduled transmissions can be many times longer than the largest packet supported by the underlying transmission medium. Since we assume that the aperiodic reserve will always be present and since we have total freedom in determining its period and average request size, we can fragment the long transmissions with an aperiodic reserve whose period is shorter than every other entry in the stream set. The second benefit has to do with receiving state information from hosts that are currently not transmitting any periodic streams. If the scheduler does not probe these hosts to allow them to make aperiodic transmissions, then the scheduler would have no way of knowing if these hosts want to create new periodic streams.

### 4.4.5 Token Based Operation

The basic operation of the scheduler consists of a loop that has four steps: (i) the scheduler creates a token for the current event, (ii) the token is sent to the associated host, (iii) the scheduler waits for the transmission of a data packet whose '`last`' field is set (this is the sign of the completion of the current event), and finally (iv) the scheduler advances to the next event of the schedule. Figure 4.6(a) shows the packets created by a three event segment of a schedule. The apparent bottleneck here is that the determination of the completion of the current event requires the scheduler to operate in promiscuous mode.

An alternate arrangement is to have each host send the token back to the scheduler. Though the need for the promiscuous modes is eliminated, each data transmission now must be preceded by two token passes; figure 4.6(b) shows the changes caused by this modification.

Although not implemented in the current version of the protocol, a third token passing pattern that has the strengths of the above schemes is based on the idea of loading several events worth of information on the token. That is, instead of the scheduler creating tokens



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

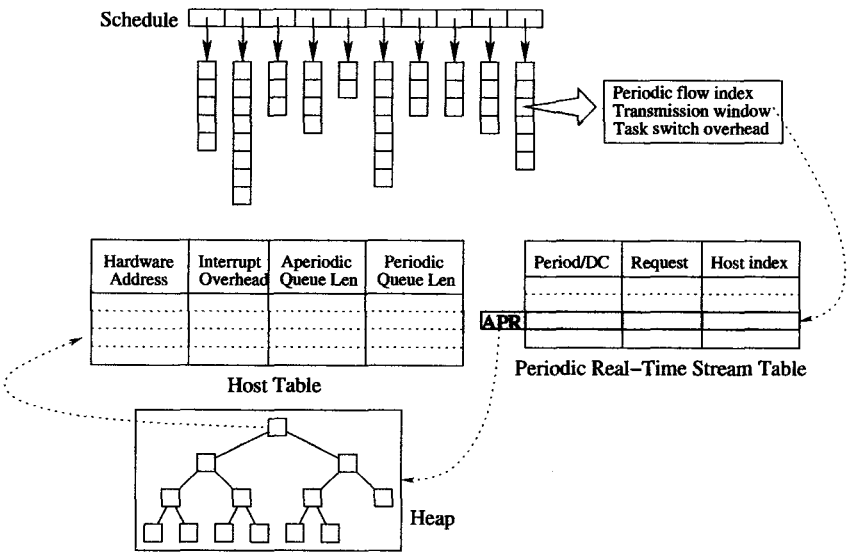



Figure 4.5: APERIODIC TRANSMISSIONS: *Since the aperiodic reserve is just a place-holder in the host table (as opposed to an actual periodic real-time stream), its events associate with actual hosts only at the instant those events are used (i.e. after the schedule has been constructed). Here, we show the use of a priority queue as the auxiliary structure for the determination of the host for an aperiodic transmission. In the top portion, we see that an event from a particular scheduling interval associates with the aperiodic reserve (APR) from the periodic real-time stream table. The system then uses a priority queue to determine the host that most urgently needs the transmission of an aperiodic packet. This host is then used to update the "host interrupt overhead" value of the event to determine the true transmission window which the leads to the generation and transmission of a token.*

specific to particular events, the scheduler creates tokens that carry $n$ events worth of source/type/duration triplets; figure 4.6(b) shows the case for $n$=2. This way, each event is preceded by one token and the rate (and burden) of token generation on the scheduler is reduced. The trade-off with this approach is that as $n$ increases, the ability of the scheduler to inject unscheduled events into the normal event stream is reduced. As we will see in chapters 5 and 6, these unscheduled events are one of the primary ways in which messages that are larger than the expected value are transmitted without causing deadline failures elsewhere in the system.

In each of the three cases, a non-trivial portion of the bandwidth is used by the token. In



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

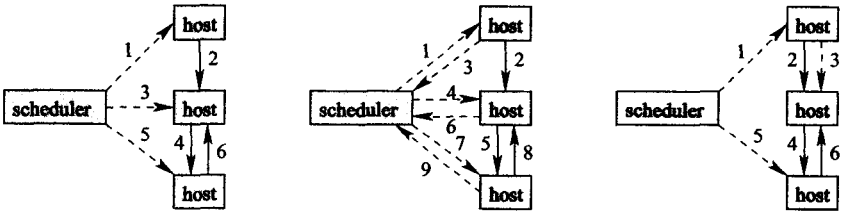



Figure 4.6: ALTERNATIVE TOKEN PASSING SCHEMES: *Part 4.6(a) illustrates the sequence of transmissions for a three-event segment of the schedule. The dashed arrows represent the token, solid arrows represent data packets, and the numbers next to arrows indicate the sequence of events. Here, each event requires the scheduler to send a token to the host associated with the current event. In part 4.6(b) the required traffic for the same three events is increased since the hosts are now expected to send the token back to the scheduler when their transmission is complete. Part 4.6(c) shows a third variation where two (instead of a single) events are loaded on the token; hence the transmission of the second token (labeled with a 3) from a host rather than the scheduler. This setup has the advantage of reducing the token generation rate imposed on the scheduler but also has the cost of reducing the responsiveness of the scheduler as the number of events loaded on the token is increased.*

addition, the scheduler carries a significant burden in generating these tokens and waiting for the completion of the corresponding events. However, a "token-dense" mechanism also has several strengths. First, the discovery of a host failure can be very rapid (this is true whether the failed host is the scheduler or not). In addition, the short lived tokens eliminate the need for complex recovery procedures often found in protocols to restore a lost/corrupt token. This in turn provides the fast response needed given the delivery deadlines of the messages.

### 4.4.6 So, What is Missing?

In this section, we outline a number of features that an actual network protocol would have to provide in order to be fully operational. Our protocol, on the other hand, only has the hooks for these features since our primary goal was algorithmic and probabilistic considerations of real-time scheduling techniques:



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- SEGMENT RECONFIGURATIONS: If the protocol is to accommodate hosts that arrive after the real-time mode starts, the scheduler should then periodically broadcast "join if you want" messages. If only a single host is waiting to join, then it can be trivially worked into the schedule to receive the token as part of the aperiodic transmissions (after which it can make admissions tests). If, however, multiple hosts are waiting, a collision would occur and the new hosts would employ the exponential back-off technique, contained to the "join if you want" slots. When the scheduler senses the collision, it increases its rate of "join if you want" messages to reduce the interval of time it takes to accommodate the new hosts.

- ERROR HANDLING: Conventional Automatic Repeat Request (ARQ) protocols solve the packet loss/corruption problem by directing the sender to retransmit packets that are not successfully delivered (in general, packet corruption is easier to deal with since the receiver can immediately detect the presence of incorrect bits). The retransmission requires the delivery of a negative ACKnowledgement in the reverse direction which is then followed by a duplicate of the corrupted data. Regardless of how rapid these steps occur, a significant amount of time passes before the complete flow of the traffic is reestablished. When we deal with real-time traffic, it is reasonable to assume that this additional time will go a significant way in causing the deadline of the packet to be missed. This situation is more extreme when packets are lost and the recovery process can begin only after specific timers expire. Therefore, for real-time traffic alone, the best policy for lost/corrupt packets may in fact be to ignore them altogether.

  With respect to hosts going down, it is possible to augment the scheduler to initiate a reaction based on a host specific timer (i.e. the marking of the appropriate row within the host table in order to reassign the bursts of the failed host). But the situation gets more complicated if it is the scheduler host that fails. In this case, the only option may be for the remaining hosts to fall back to the CSMA mode, elect a new scheduler, and switch back to the real-time mode. Again, for deadline related reasons,



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

there would be no value in buffering the real-time messages during the recovery phase.

## 4.5 Conclusion

Our primary purpose has so far been the explanation of the simulation environment and a description of the proposed network protocol. We now offer a few general remarks on algorithmic choices as we segue into chapters 5 and 6 where we report on the results of out experiments.

When all other factors are equal, dynamic scheduling algorithms are usually preferred over static ones since dynamic algorithms can theoretically achieve 100% utilization of the shared resource. However, this potential can be reached only under two ideal conditions: instantaneous knowledge of the tasks of the system and zero task switch overhead. In a network, neither of these conditions hold. In addition, the descriptive parameters such as the ready times and sizes of the messages vary around a mean as opposed to being constant throughout the life of a schedule. This makes it unlikely at best for dynamic algorithms to show their edge. Static scheduling algorithms also depend on state information but the granularity of this information can be more coarse since priorities are not modified per job. In addition, static scheduling systems are easier to implement.

The two algorithms we chose to drive the network protocol are RMS ([LL73], [LSD89]) and DCS ([HLL95], [HLF95], [HL92], [HL89]). RMS has been shown to be the optimal static scheduling algorithms. DCS, on the other hand, offers the similar worst case performance but provides a design that does not require the scheduler to keep synchronicity between its task set and the actual periodic real-time streams..



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# Chapter 5

# Rate Monotonic Scheduler and Results

The evaluation of a real-time scheduling algorithm consists of two stages. The first stage of schedulability analysis was covered in chapter 3. With the definition of a real-time network protocol and a simulation environment for experimentation, we now start the second stage of our evaluation which is an investigation of the characteristics of the produced schedules. In this chapter, we focus on RMS; in chapter 6, we focus on DCS.

We start with an overview of the design of the rate monotonic scheduler; specifically, we focus on the scheduler's method of keeping the schedule synchronized with the network events and of dealing with messages whose sizes vary around an average (section 5.1). We then present the jitter characteristics of the produced schedules (section 5.2) followed by the scheduler's ability to manage varying message sizes (section 5.3), and varying message/stream ready times (section 5.4).

Before we discuss design details, it may be useful to recap the overall operation of the scheduler as defined in chapter 4. Namely, the real-time mode is started when one of the hosts in the virtual network initiates a periodic stream and lasts until all the periodic streams end. The introduction of each periodic stream initiates an admission test during which the system attempts to create a new schedule for the new task set. If successful, the system



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

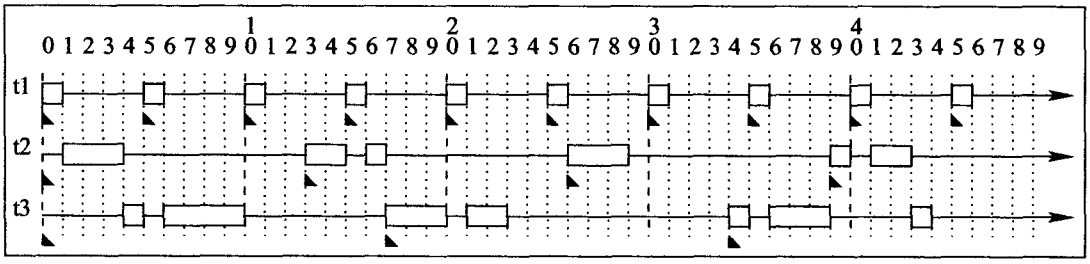



Figure 5.1: CONSTRUCTION OF A RATE MONOTONIC SCHEDULE: *A real-time task in rate monotonic scheduling is represented with a pair of numbers $\tau = (c, p)$ where p is the period of the task and c is the size of the request it makes during each of its cycles. As defined in chapter 2, the scheduling priority of a task in RMS is inversely proportional to its period: the shorter the period, the higher the assigned priority. Thus, for example, given the three tasks $(5, 17)$, $(1, 5)$, and $(3, 13)$, RMS rearranges them in a descending priority order of $\tau_1 = (1, 5)$, $\tau_2 = (3, 13)$, and $\tau_3 = (5, 17)$. As the tasks run, the scheduler always makes the shared resource available for the ready task with the highest priority. The first 50 time units of this operation is shown above. The triangles represent job ready times and the rectangular boxes on the timelines represent resource use.*

switches to the new schedule; otherwise, the new stream is rejected and the operation of the current schedule continues. The scheduler maintains host specific task switch overhead values which get used during the creation as well as the use of the schedule. These overhead values change over time and the scheduler is expected to adapt. The hosts that emit periodic traffic do so when they receive a token packet from the scheduler which specifies the type and maximum duration of the transmission. If the host has no data to send, it immediately flags the scheduler which generates and sends a new token to the next host down the line.

## 5.1 The Design of the Rate Monotonic Scheduler

The basic construction of a rate monotonic schedule by our network protocol is similar to the method outlined in [LL73]; figure 5.1 provides an example involving three tasks. One difference is that our scheduler incorporates the token passing overhead into the process and thus may reject task sets that could have been schedulable had the task switch cost been 0. A second difference is that the given task periods and the resource requests are assumed to be average values; [LL73] assumes them to be worst case estimates.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

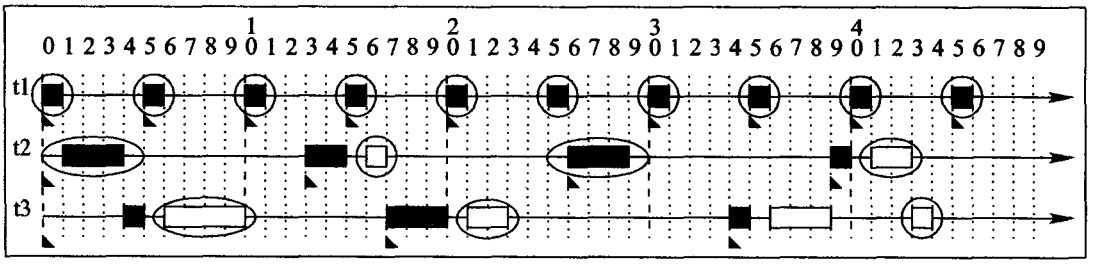



Figure 5.2: EVENT TYPES OF A RATE MONOTONIC SCHEDULE: *A rate monotonic schedule can have four types of events:* START/END *which are shaded and circled,* SYNC *which are shaded only,* END *which are circled only, and the remaining ones that are simply continuations (but not terminations) of jobs that have already started. The* SYNC *events could have been called* START *events too but we preferred the former term since these events do have a synchronization purpose as will be described soon.*

When we consider the activities of the partial schedule shown in figure 5.1, we see four types of events. The first type, which we refer to as SYNC events, represents the start of new jobs. The second type, which we refer to as END events, represents the completion of jobs. In figure 5.2, the SYNC events are shown as shaded rectangular blocks while END events are circled rectangular blocks. Since these two sets are not mutually exclusive, some of the events (notably all of the ones of $\tau_1$) belong to both. Finally, there are those events that represent the continuation but not the completion of jobs; these are neither shaded nor circled; there is only one of these and it is within the third job of the third task.

A rate monotonic schedule constructed by our data-link protocol is arranged in terms of **scheduling intervals**. Each scheduling interval starts with a SYNC event and continues up to the first END event; figure 5.3 illustrates the scheduling intervals of the running example. In this simple schedule, many of the scheduling intervals consist of a single event. Those that have more than one are connected by arrows, with the first arrow rooted in the starting SYNC event.

The operation of the scheduler is centered around the processing of the scheduling intervals. There are two primary reasons for this:

- SYNCHRONIZATION BETWEEN THE SCHEDULER AND THE NETWORK: Rate monotonic scheduling is based on the idea of providing the resource precisely at the instant it is



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

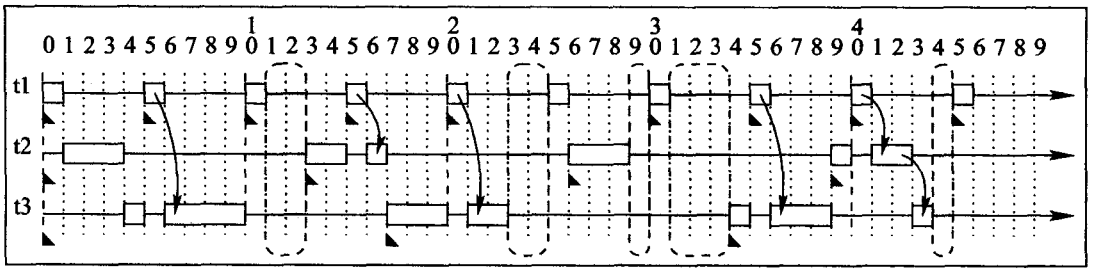



Figure 5.3: SCHEDULING INTERVALS OF A RATE MONOTONIC SCHEDULE: *Scheduling intervals form the coarse granularity of events within a schedule. In the simplest case, a scheduling interval consists of a single* SYNC *event. As the schedules get more elaborate, however, many hold multiple events. Here, the members of multi-event scheduling intervals are linked with arrows. The diagram also shows the idle gaps (shaded vertical rectangles spanning the three timelines) that get used when transmitting packets that are larger than expected. The average number of events in a critical region depend on the particular resource request and period values of the task sets. In general, the longest-period/shortest-period ratio and request-period correlation (if there is one) are also important factors.*

needed. Therefore, providing the resource more often than needed or ahead of time does not necessarily improve the performance of a system; in fact, it could make things worse. For example, if the first three transmissions in figure 5.1 turn out to be shorter than expected, then a scheduler operating without any synchronization mechanism sends the token for the fourth event (i.e. the second job of the first task) earlier than anticipated. Without a waiting message, the host decides to return the token back to the scheduler and misses its opportunity to transmit the second message.

A second reason for synchronization is the reduction of the scheduling overhead on the scheduler. As outlined in the previous chapter, the scheduler performs an O(1) operation during each scheduling event. Although this is as good as it gets in terms of asymptotic complexity, the actual amount of code executed per event is non-trivial. When events last shorter than expected, the token is passed around at a rate that is faster than what has been established as sufficient during the schedule construction. The scheduler then runs the risk of falling behind, itself becoming a reason of deadline failures.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The basic token handling rule for the hosts in the network is to make a transmission as soon as the token is received; if the token receiver has no message to transmit, then it is required to relinquish its turn. The scheduler solves the synchronization problem with a simple exception to this rule: when a host receives a token that is marked with the SYNC flag, it is permitted to wait for the generation of a message if there is none already waiting. This allows the scheduler and the network activities to proceed in lock step. Since the synchronization is performed only at the SYNC events, the scheduler also maintains some freedom to stretch the boundaries of non-SYNC events in the associated scheduling intervals. This, as we will see further down, provides some capacity to transmit messages that are larger than the time allotted for them.

- MANAGEMENT OF IDLE TIME: As long as the total utilization of a task set is less than 100%, the corresponding schedule contains scattered gaps of varying lengths that could be used either to increase the durations of the neighboring events or for squeezing in unscheduled (i.e. new) events. As the schedule is used, additional gaps also appear due to events that take shorter than their anticipated duration and to periodic streams that terminate. Real time schedulers vary a great deal in terms of making use of this idle time. Our protocol uses it to increase the transmission windows of existing events. There is, however, one exception to this which will be described shortly.

  Much like the synchronization process, the management and utilization of the idle time is arranged around scheduling intervals. The basic idea is that as the system enters a new scheduling interval, a state variable called TIMECREDIT is set to the amount of free time available. The scheduler then steps through the events within that interval, increasing the transmission window of the current event either by the entire value of TIMECREDIT or by an even share of it (the choice depends on a protocol configuration switch); the pseudo-code for both schemes are given in figures 5.4(a) and 5.4(b). In both cases, time gained by events that last shorter than expected or by the events of



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
credit = free time in new interval
SYNC event transmission window += ( credit / num of events in interval )
credit -= ( credit / num of events in interval )
execute SYNC event
credit += anticipated time of SYNC event - actual time of SYNC event
while ( there are remaining events in the current interval )
    current event transmission window += ( credit / num of events in interval )
    execute current event
    credit += anticipated time of current event - actual time of SYNC event
```

(a) Idle time management: CREDITALL

```
credit = free time in new interval
SYNC event transmission window += ( credit / num of events in interval )
credit -= ( credit / num of events in interval )
execute SYNC event
credit += anticipated time of SYNC event - actual time of SYNC event
while ( there are remaining events in the current interval )
    current event transmission window += ( credit / num of events in interval )
    execute current event
    credit += anticipated time of current event - actual time of SYNC event
```

(b) Idle time management: CREDITEVEN

Figure 5.4: IDLE TIME MANAGEMENT SCHEMES IN THE RATE MONOTONIC NETWORK SCHEDULER: *In each scheduling interval, current event transmission is either increased by the entire amount of accumulated idle time (pseudo code in part 5.4(a)) or by an even share of it (pseudo code in part 5.4(b)).*

tasks that have terminated are added to TIMECREDIT for the benefit of the remaining events.

As the system enters a new scheduling interval, setting the value of TIMECREDIT with the size of a potential idle gap consists of a single lookup. Similarly, when executing the events within that interval, updating the value of TIMECREDIT with the transmission windows of terminated periodic streams are trivial. The detection of the time gained by events that last shorter than expected, however, requires the scheduler to measure event-durations. Based on the host-overhead model discussed in chapter 4, these measurements require starting a timer at the instant the scheduler creates the token and stopping it at the instant the transmission of the corresponding messages are complete. For non-SYNC events,



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

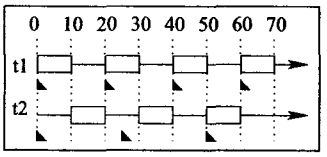



Figure 5.5: THE IRREGULARITY OF THE WAIT TIMES UNDER RATE MONOTONIC SCHEDULING: *As the first 70 times units of the schedule for tasks* $\tau_1 = (20, 40)$ *and* $\tau_2 = (20, 50)$ *show, the first, second, and third jobs of* $\tau_2$ *have to wait 20, 10, and 0 time units, respectively.*

the timer value minus the associated overhead is all that the scheduler needs. For SYNC events, however, the timing process is more complex. As stated earlier, SYNC events allow the receiver of the token to wait for the generation of a new message but this permission is not always put to use; an example of the consequential irregularity even from a simple task set is shown in figure 5.5.

With periods that do not align well and with token passing overheads that are host specific, the scheduler cannot readily determine what amount of time should be deduced from the observed event interval as it updates TIMECREDIT. A solution, however, is available during the schedule construction stage. For each SYNC event, the scheduler keeps track of the time difference between the ready time of the corresponding message and the scheduled time for its transmission. If this interval is beyond a threshold level close to zero, the expected delay is subtracted from the observed event duration; otherwise, no subtraction is performed.

The expected wait associated with SYNC events is put to use in a second way. If the scheduler estimates that the wait will be greater than a minimum-data-packet-size related threshold, then it allows that host to make an unscheduled transmission during its wait. The overhead cost of this unscheduled event is zero, since the host will already be getting the token. In addition, it offers hosts a last chance to complete the transmissions of partially sent messages which, due to the real-time requirements of the network traffic, will otherwise be dropped at the instant the new message is created; the details of the decisions behind this particular operation is outlined in figure 5.6.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

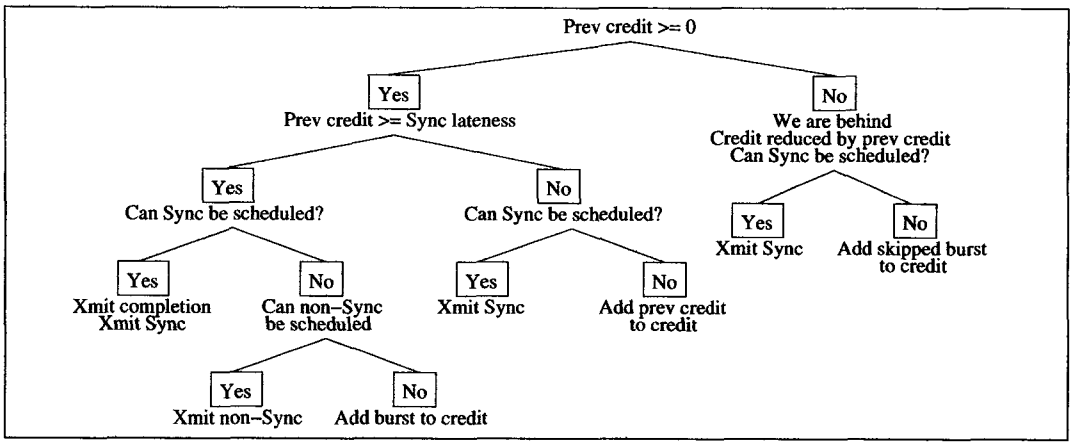



Figure 5.6: IDLE TIME MANAGEMENT IN THE RATE MONOTONIC NETWORK SCHEDULER: *The conditions considered by the scheduler as it moves from one scheduling interval to the next one is shown above. "Credit" refers to the free time found in the current busy interval while "prev credit" refers to the unused time inherited from the previous one.*

We end this section by addressing several issues that relate to the design and operation of the rate monotonic scheduler:

- WHAT HAPPENS TO THE SCHEDULED BURSTS OF A TERMINATING PERIODIC STREAM? Since the process of creating a new schedule is computationally expensive, our protocol does not create a new one when a periodic stream terminates. Instead, the events corresponding to this periodic stream are used to increase the transmission windows of the other events that fall within the same scheduling intervals. If an event of a terminated periodic stream is the only one in its scheduling interval, then its transmission window gets inherited by the following scheduling interval in the form of an unscheduled event.

- WHY IS THE IDLE TIME IN A SCHEDULING INTERVAL CONSIDERED ONLY FOR LENGTHENING THE SCHEDULED TRANSMISSION AND NOT FOR SCHEDULING NEW ONES? The are several reasons for this. First, due to the long task switch overhead that is inherent in network scheduling, the gain in scheduling a new transmission is often less than increasing the length of existing events whose overheads have already been accounted for. Second, the effectiveness of unscheduled transmissions depend on the state infor-



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

mation maintained by the scheduler. That is, if the scheduler knows the host that is in the most dire need of a transmission, then it can send the token to that host. Unfortunately, since the real-time system in this case happens to be a network of computers (as opposed to a single processor running multiple tasks as assumed by many of the studies in this field), this information is never available in a timely and reliable way. Since the hosts contain a two slot queue for real-time traffic, the scheduler can never learn and react quickly enough to pending deadline failures. In fact, if it could have reacted fast enough, then the entire reason behind proposing a static schedule would have to be thrown out in favor of dynamic scheduling algorithms which provably have better utilization rates. A third (and somewhat more morbid) reason is that, as will be shown at the end of this project, RMS has various characteristics that render it less than ideal for network scheduling (at least in the context we considered it). Therefore, even if we would have looked at the statistical properties of the sources and decided in advance where the token should be sent when sufficiently long gaps form, augmenting the protocol in this manner would have been fruitless in the grand scheme of things.

- IS IT POSSIBLE FOR AN EVENT TO LAST LONGER THAN EXPECTED? The upper limit specified in a token should ordinarily prevent this. But as the system evolves, the responsiveness of the hosts change. Therefore, it is possible for some of the events to take longer than anticipated only because the host receiving the token is overloaded and is not able to complete the transmission within the scheduler's estimate. The scheduler's primary concern is to ensure that the total duration of the scheduling interval remains constant. Therefore, it "regrettably" cuts down the transmission window of the *next* event by the required amount to offset the loss. However, this unfairness does not continue beyond a single event since the scheduler also updates its overhead estimate of the slow host and thus starts to cut its net transmission windows from future events.

- DOES THE NETWORK PROTOCOL QUEUE MESSAGES AT ALL? Due to the real-time



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

nature of the problem, we configured the protocol to operate with a single slot queue. This way, as a message is being transmitted, the next one can be assembled by the application using the protocol. Technically, we could increase this queue size to two (or higher) to reduce the message drop rates. This would, however, cause a high rate of deadline failures. In fact, if a $q$ slot queue is used, then the upper bound for the transmission time for a periodic stream would be $q$ times the period of that stream.

- DURING TRANSIENT PERIODS OF OVERLOAD, WHICH MESSAGES GET DROPPED? If the application starts generating a message when the single slot queue is full *and* the transmission of the previous message is not complete, then the system drops the partially transmitted message. Initially, this seems to be the wrong choice. After all, some amount of time has already been invested into transmitting this message and the system might as well try to complete its transmission, dropping the queued message instead. However, this choice leads to increased latency during periods of overload; that is, messages get delivered to their destination beyond the end of the period in which they were generated. Given the real-time nature of the context, slight underutilization of the bandwidth is preferable over messages delivered late.

## 5.2 Jitter Characteristics

Once the implementation of the data-link protocol outlined in the previous section was complete, we ran a range of simulations to study its operational characteristics. Since the effectiveness of RMS within simple computational models has already been established, our focus was on studying the performance of this algorithm when the underlying system had a large number of parameters (such as the sizes of the requests, the start times of the periodic streams, the responsiveness of the hosts, etc) that did not remain constant over long periods of time. In this section, we focus on the job completion time jitter of the produced schedules.

**Interarrival time** represents the time interval between a pair of consecutively received messages. **Jitter** is the standard deviation of the interarrival times observed over a stream of



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

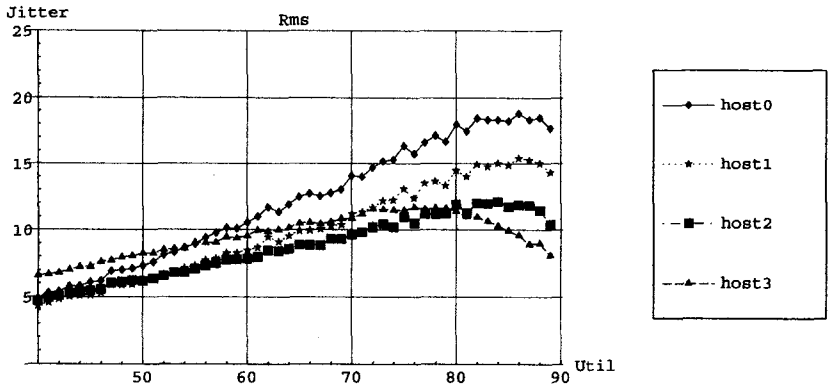



Figure 5.7: JITTER CHARACTERISTICS OF RATE MONOTONIC SCHEDULING: *During this experiment, the target task set utilization (x-axis) ranged from 40% to 90%. At each point, 250 task sets were randomly generated whose total utilizations were within half a percentage point of the target. A rate monotonic schedule was then constructed for each task set and the jitter characteristics of the resulting events were computed. In order to keep the results independent of the underlying task sets, the jitter values of a task were normalized with the period of that task. The final numbers were therefore expressed as percentage values on the vertical axis. Finally, the average of the 250 normalized jitter values was plotted for each task at each target utilization.*

messages. In just about every system that deals with periodic traffic, the reduction of jitter (or even elimination, though usually not possible) is an important design goal. Therefore, it seemed appropriate to start the simulations by looking into the jitter characteristics of the produced schedules. This first set of results is plotted in figure 5.7.

When we look at the slopes of the curves in figure 5.7, we see the combined effect of two factors that are often at play within real-time scheduling. The first is related to the fact that as the stream set utilization goes up, the number and sizes of the idle gaps that exist in the schedule for new events are reduced. This in turn increases the likelihood that an event will be scheduled further down the line than where it would have been if the system had no other streams. We can see the effects of this factor within the 40%-80% interval of the stream set utilization where the jitter measure for all sources increases in an almost linear manner. However, we say "related to" instead of "determined by" since jitter is a measure of the *variations* in these scheduling delays rather than their mean. The increase specifically has to do with the *non-uniform* nature of the reduction of the sizes and numbers



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of the idle intervals. This turns out to be the general tendency of the combined scheduling requests of streams whose periods are not aligned.

However, once we get past the 80% utilization level of, we see a change. Beyond this point, it appears as though the normalized jitter rates go down but this has little to do with any improvement in the characteristics of the underlying scheduling algorithm. At high utilization levels, stream sets are more likely to be rejected by the scheduler if their periods are closer to being relatively prime. Therefore, at higher utilization levels, the jitter experiments are being based on a greater proportion of streams whose periods are close to being aligned. Better period alignment leads to more regular spacing of the idle gaps. This in turn leads to more evenly distributed separations between the ideal and the actual scheduling instants of the events which finally reduces the variations of the interarrival times of the messages. In fact, based on this observation, we can speculate that only perfectly aligned stream sets can be scheduled at the extreme case of 100% utilization. This is consistent with a jitter value of 0 to which the curves seem to be headed.

We also see that there is an inverse correlation between the size of the distribution and the priority of the task. This pattern emerged in many of the experiments, reflecting the fact that the effect of the assigned priorities in real-time scheduling systems is limited to temporal correctness without necessarily leading/relating to any other type of advantage (such as, in this case, regularity of completion times).

## 5.3 Message Size Variations

The jitter related results of the previous section were collected under conditions that remained constant until the end of the simulations. That is, once the schedules were generated, there was no variation in the message sizes, message generation times, or stream start times. Actual networks, of course, are complex systems and seldom remain in a constant state for non-trivial intervals of time. In order to gauge the performance of the rate monotonic network scheduler in more realistic situations, we conducted a series of simulations where the



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

characteristics of the underlying system was varied in ways we expect from real networks.

In this section, our focus is on observing the consequences of allowing the hosts to generate messages that vary around the averages used during the construction of the schedule. In each of the simulations, the message drop rates as well as the message delivery times are recorded. In order to focus on the characteristics of the algorithm, the network is configured not to lose or corrupt messages. Since the separate issue of schedulability was studied in chapter 3, the tested periodic stream sets are all schedulable.

In figure 5.9(a), we see the message drop rate results for periodic stream sets with four members $\{\tau_0, \tau_1, \tau_2, \tau_3\}$. As in the schedulability plots, one of the independent variables is the stream set utilization (i.e. includes the total bandwidth required by all the streams). The second independent variable is the standard deviation that determines the message size variations. In order to produce results that are number-independent, the standard deviation is specified as a percentage of the expected message size. For example, a value of 10% on this dimension with an expected message size of 500 means that the message size distribution for that stream has a standard deviation of 50.

As expected, RMS has no message loss when the size distribution has a standard deviation of 0. But when this parameter is raised to just 1%, a noticable level of messages are lost, though with a magnitude that is reduced for lower priority tasks. Looking at the slopes of the surface as we go toward higher variances, we see that the scheduler is using its idle-gap capacity well at lower utilizations. At higher utilizations, the scheduler has to work with a small number of idle gaps and thus cannot usually accommodate excess requests.

Although a surface plot of an average is an efficient summarization, it reveals nothing about the variance within the data. For this reason, we plotted two dimensional "slices" of the surface in figure 5.9(a) where each slice represents one of the twenty percentage levels used in setting the message size variance. The slices for task $\tau_0$ of figure 5.9(a) are in 5.10(a); likewise, slices for tasks $\tau_1$, $\tau_2$, $\tau_3$ are in figures 5.10(b), 5.10(c), and 5.10(d), respectively. Since the two dimensional plots are too small for the axis labels and ticks to be read (our goal is just to show trends), we included a "zoomed up" sample in figure 5.8. The error



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

bars represent one standard deviation within the data.

We observe the higher priority tasks suffer a greater variance in message loss as well; for example the error bars in 5.10(a) are much greater than those in 5.10(d). The main reason for this has to do with the tasks with long periods also having in general longer messages to send. Therefore, when the message size of a low priority task is increased by, for example, 10%, it causes a significant impact on tasks with sorter periods (which happen to be the tasks with higher priorities). If the percentage based message size variances had been based off of a fixed parameter for all task (rather than each tasks mean message size) or if longer periods had not positively correlate with longer requests, then higher priority tasks would not have been disproportionately effected.

Figure 5.9(b) shows the message delivery times which represent the duration from the instant a message is generated to the instant it is delivered (i.e. the sum of the wait at the source host, the host overhead for initiating the transmission, and the actual transmission itself). Again, in order to have stream set independent data, we report this interval after normalizing it with the period of the associated stream. Here we see that higher priority tasks get their message across more quickly than lower priority tasks. Even as the stream set utilizations reaches 90%, the messages of the highest priority stream reach the destination host after 40% of their period. At the same utilization level, the same duration for the lowest priority stream is close to 60%. We also note that the increase in the message size variance does not have much of an effect on the message reception time (i.e. the surface height does not increase much as we go to higher values along the "ReqSD" dimension). This is because once the scheduler realizes that a message cannot be transmitted without causing deadline failures, it discards that message.

In chapter 3, we showed two ways in which the periods of randomly generated tasks are drawn: the uniform distribution vs the indirect uniform distribution. We also stated that the task generator could be configured to impose a positive, a negative, or a zero correlation between the message sizes and the associated periods. The effects of these factor are explored in our simulations as well. In figure 5.12 (where we show message loss rate once more), all



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

plots are based on uniform period distribution. In the surface plots of figure 5.12(a), there is a negative correlation between randomly generated periods and utilization value pairs while in figure 5.12(c), the correlation factor is positive (5.12(b) is a repetition of figure 5.9(a) for convenient visual comparison). In figure 5.13, on the other hand, all plots are based on indirect uniform period distribution (figures 5.13(a), 5.13(b), and 5.13(c) show the effects of negative, zero, and positive correlation between the randomly generated periods and utilization value pairs). When we compare figure 5.12 to figure 5.13, we see that the main difference seems to be between 5.12(c) and 5.13(c). Specifically, using indirect uniform distribution for randomly generated utilization values which have a positive correlation with their associated period values leads to the greatest message loss. Therefore, if additional simulations were to stress test the rate monotonic scheduler, this would be the configuration that would constitute the worst case in general. We also see that, for the highest priority periodic streams, negative correlation between the period and the utilization of a periodic task causes more message loss than a positive correlation; for all the other tasks, the reverse is true.

We finally explore the effects of the two idle time management schemes we outlined in section 5.1. In figure 5.14(a) (which is a repetition of figure 5.9(a) for convenient comparison), idle time is managed according to the CREDITALL scheme of figure 5.4(a) (assigns all of the idle time available in a scheduling interval to the current event). Figure 5.14(b) shows the *additional* message loss rate once we switch from the CREDITEVEN scheme of figure 5.4(b) (evenly divides all of the idle time available in a scheduling interval between its member events). Since we see either no change or a positive change (i.e. the message loss rate either remains the same or increases), we conclude that the CREDITALL scheme is more effective that the CREDITEVEN scheme for managing idle time. Indeed, additional experiments (whose plots are not included in this writeup) confirmed the conclusion that, on average, it is better to hand all of the idle time to the current event as opposed to sharing it evenly. Consequently, the remaining experiments shown in this chapter are all based on the first scheme.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

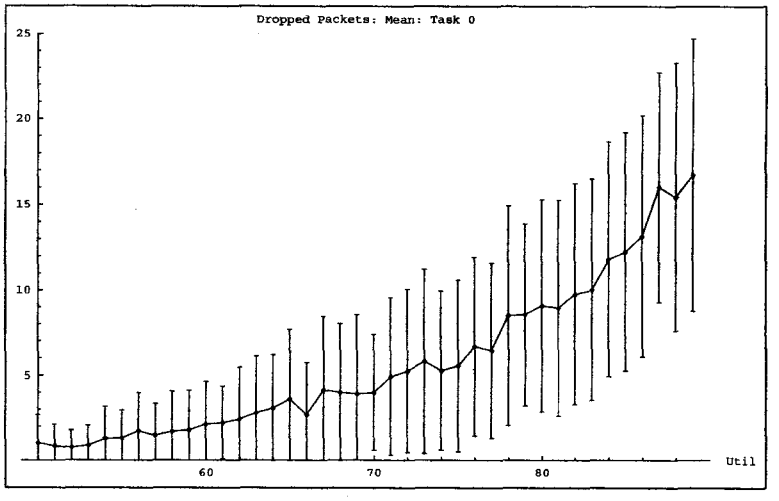



Figure 5.8: DETAILS OF THE "2-D SLICE" PLOTS USED IN CHAPTERS 5 AND 6: *The variance hidden in the surface plots are shown in several full page figures where the three dimensional data is arranged in "slices" (e.g. figure 5.10). In order to display as many slices as possible, these two dimensional plots are reduced in size so much that reading the numbers or the labels is no longer possible. We think that this is OK since our goal is to show trends. Here, however, we present one sample (in larger size) just so that the reader can see the lost fine print. In every case, the solid curve that spans the utilization axis represents a mean; the error bars around the mean represent the size of a single standard deviation.*

## 5.4 Message/Stream Ready Time Variations

Given the multitasking nature of the hosts, it is reasonable to assume that the periods associated with the periodic streams will be averages and that there will be minor variations in terms of the instants at which the messages are generated. The next set of experiments investigate the influence of this inaccuracy. The surface plots for message loss rates for four periodic stream sets are shown in figure 5.15(a). As before, the first independent variable represents stream set utilization (i.e. the total bandwidth required) and the vertical axis represents message lost rate. However, the second independent variable now specifies the standard deviation of the message generation time distribution (Gaussian) as a percentage of the associated period.

We see that the system is holding up well until this parameter goes up to 10% but for higher rates, message loss shoots up to unacceptable levels. This is because RMS is very



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

sensitive to timing issues since the schedules are specific to message generation times. Up to a certain point, the variations are minor enough that the system can withstand a number of them in a row; at that point, since these variations are spread around the expected values, they cancel each other. But once the variations start to exceed a threshold level (which, as we have stated above to be at around 10% in these particular simulations), then individual variations are significant enough to disrupt the operation of the scheduler and start cascading temporal errors. The average delivery times of the messages are shown in figure 5.15(b).

Again, we display the variance hidden in the surface plots as slices through these surface plots. Those for the message drop rates are displayed in figure 5.16. As we would suspect, the variance within the regions where the protocol has stopped to be useful are very high. As the last five or six slices we see in each of the four subfigures indicate, every periodic stream has effectively been broken; that is, the scheduling priorities do not insulate any of the streams from this occurrence at runtime. The variance with the transmission time data is shown in figure 5.17.

When a new schedule is created, it is usually assumed that every stream is ready to go at the instant the schedule switch occurs. However, within a network of computers, this assumption cannot readily be made and may require an explicit synchronization. The last set of experiments to gauge the performance of RMS looked into the effect of starting the periodic streams at random times. The results are shown in figures 5.18, 5.19, and 5.20. The main difference between this set of plots and the previous batch is that the second independent variable now represents the percentage of the the first period within which the start time of a periodic stream is selected. For example, if this value is 10% and we have a periodic stream whose period is 500 milliseconds, then the simulation may initiate it any time within the interval 0 to 50 milliseconds.



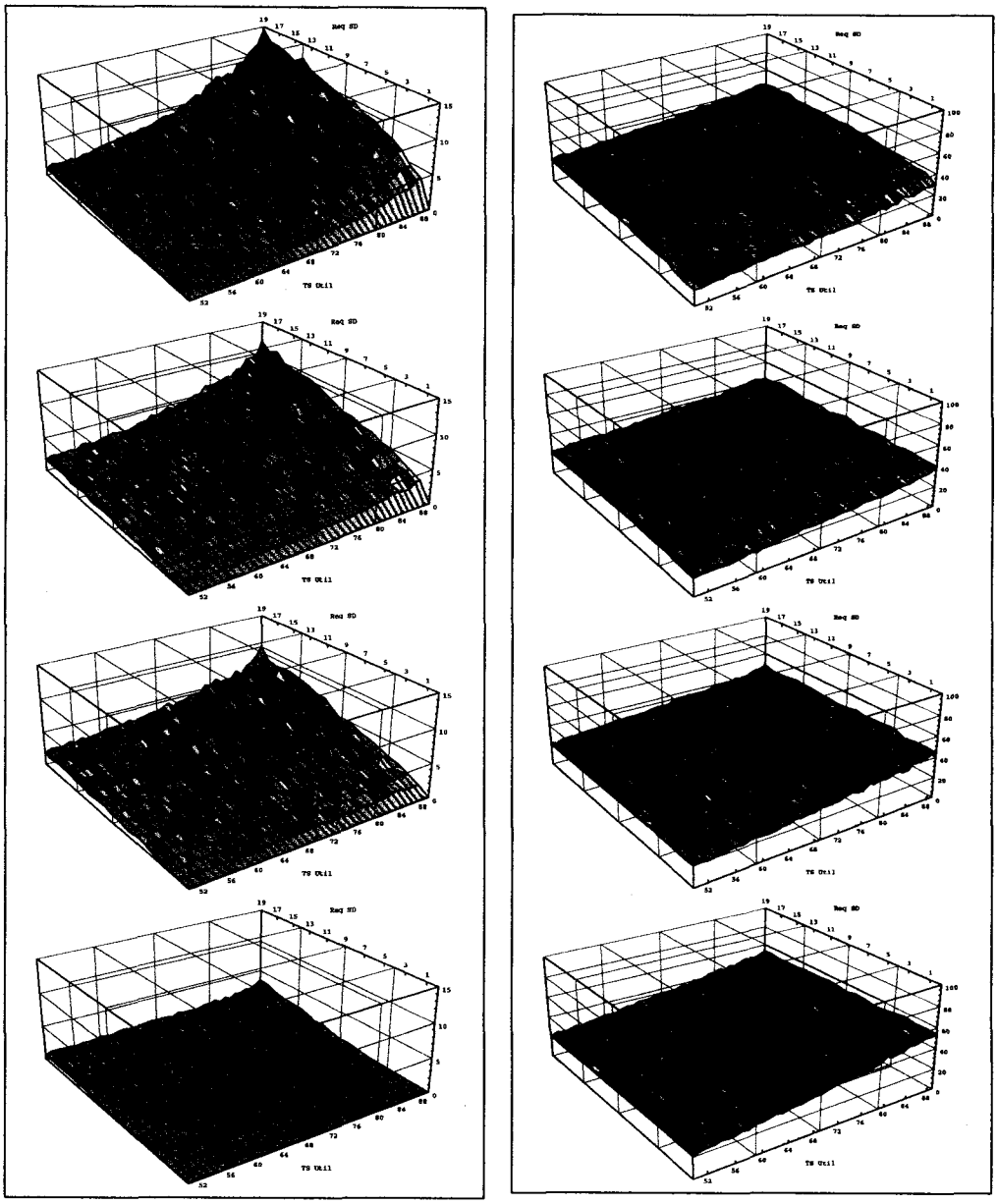
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

## 5.5 Conclusion

Chapter 5 summarized the design and results of the real-time data-link layer network protocol based on RMS. The next chapter provides the corresponding results of the distance constrained scheduler; a qualitative comparison of the two schemes is in chapter 7. We conclude this chapter by noting that we considered implementing an alternative version of the rate monotonic scheduler where the last event of every scheduling interval would be unconstrained. The motivation for this was to eliminate the token holding scheme (as determined by SYNC events) to simplify the operation of the scheduler. Since the distance constrained scheduler turned out to perform better than the rate monotonic scheduler by a margin greater than this optimization would have provided, this idea never materialized.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



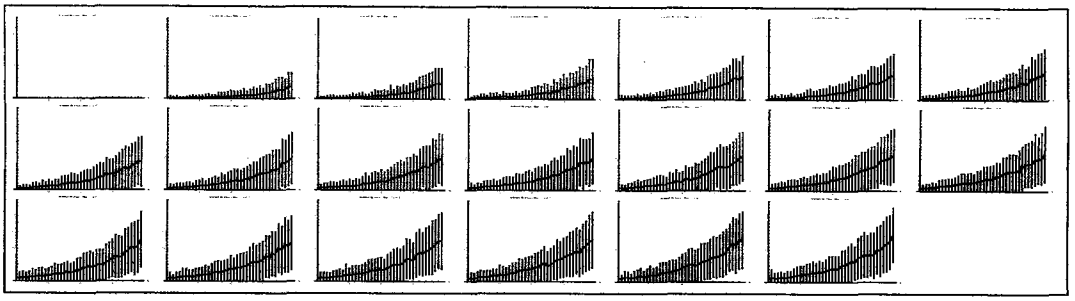
(a) Percentage of the messages dropped by RMS. The range of vertical axis is 0%-15%.

(b) Normalized message delivery times. The range of vertical axis is 0% to 100%.

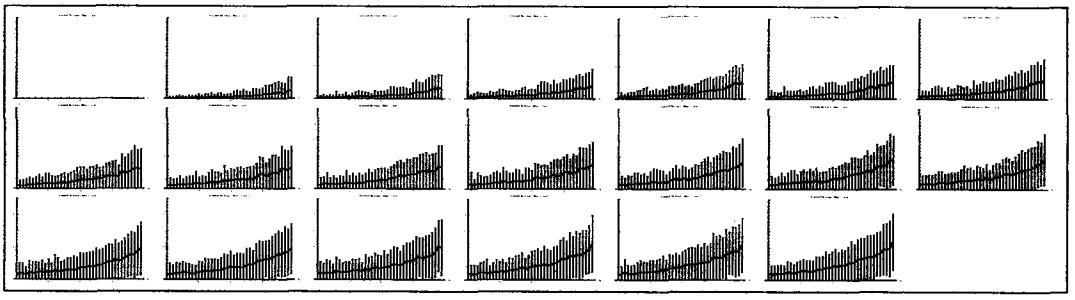
Figure 5.9: INFLUENCE OF MESSAGE SIZE VARIATIONS ON RMS MESSAGE LOSS: *For each of the plots, task utilization varies between 50%-90% and the StdDev for message size varies between 0%-20%. The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom.*



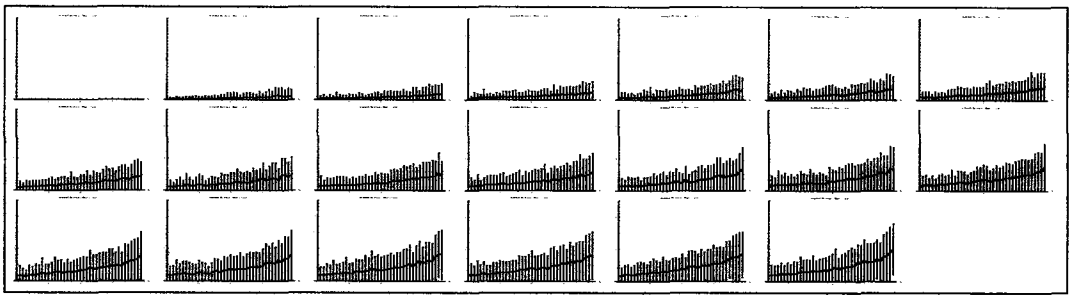
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



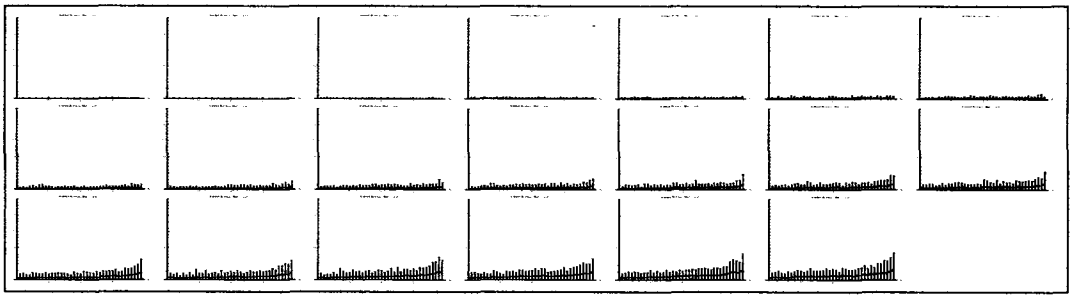
(a) Task $\tau_0$



(b) Task $\tau_1$



(c) Task $\tau_2$

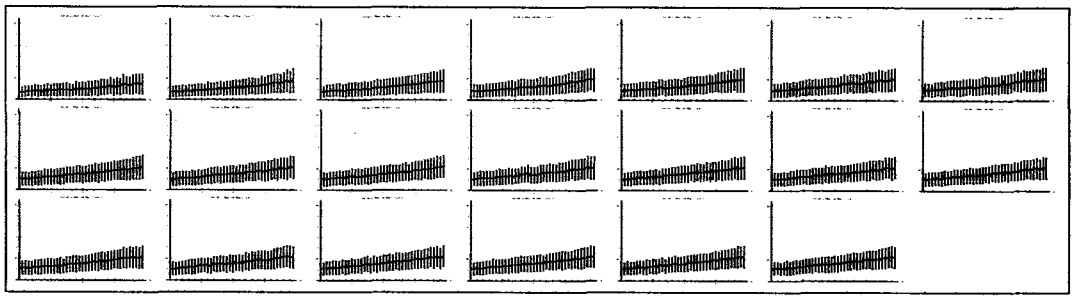


(d) Task $\tau_3$

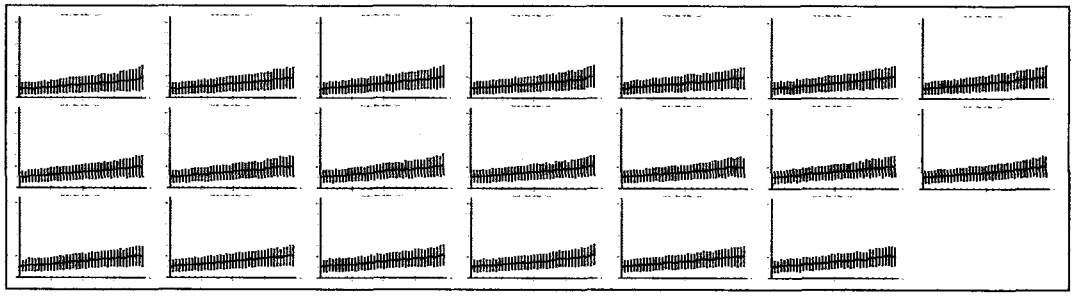
Figure 5.10: 2D Slices of surface plots in 5.9(A)



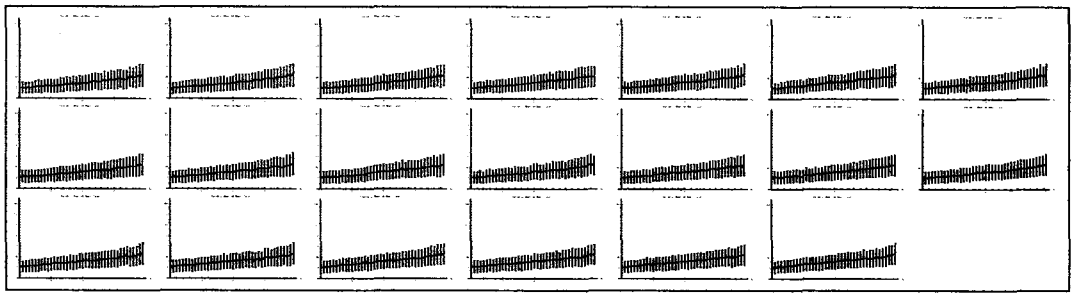
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



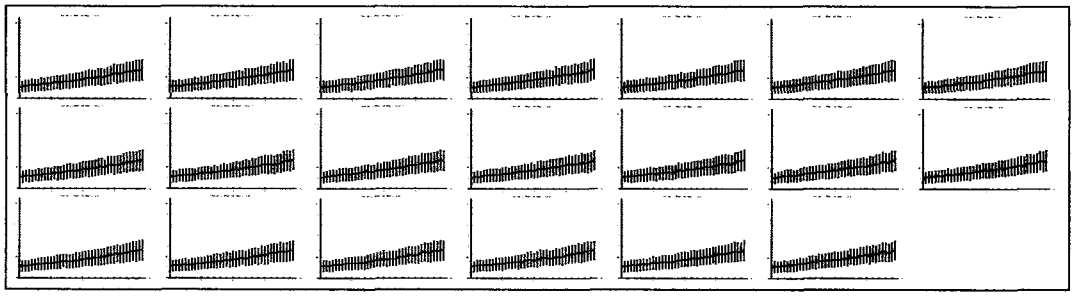
(a) Task $\tau_0$



(b) Task $\tau_1$



(c) Task $\tau_2$

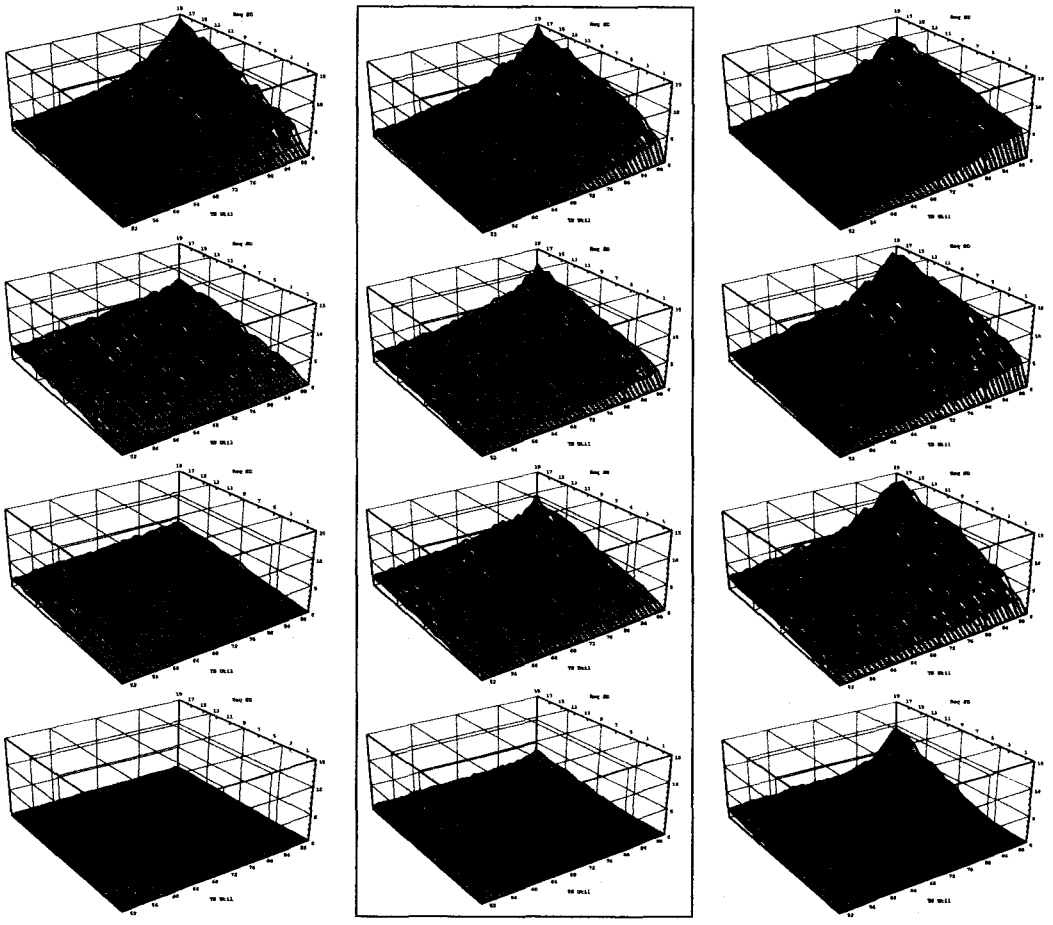


(d) Task $\tau_3$

Figure 5.11: 2D Slices of surface plots in 5.9(B)



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Negative correlation between period and utilization

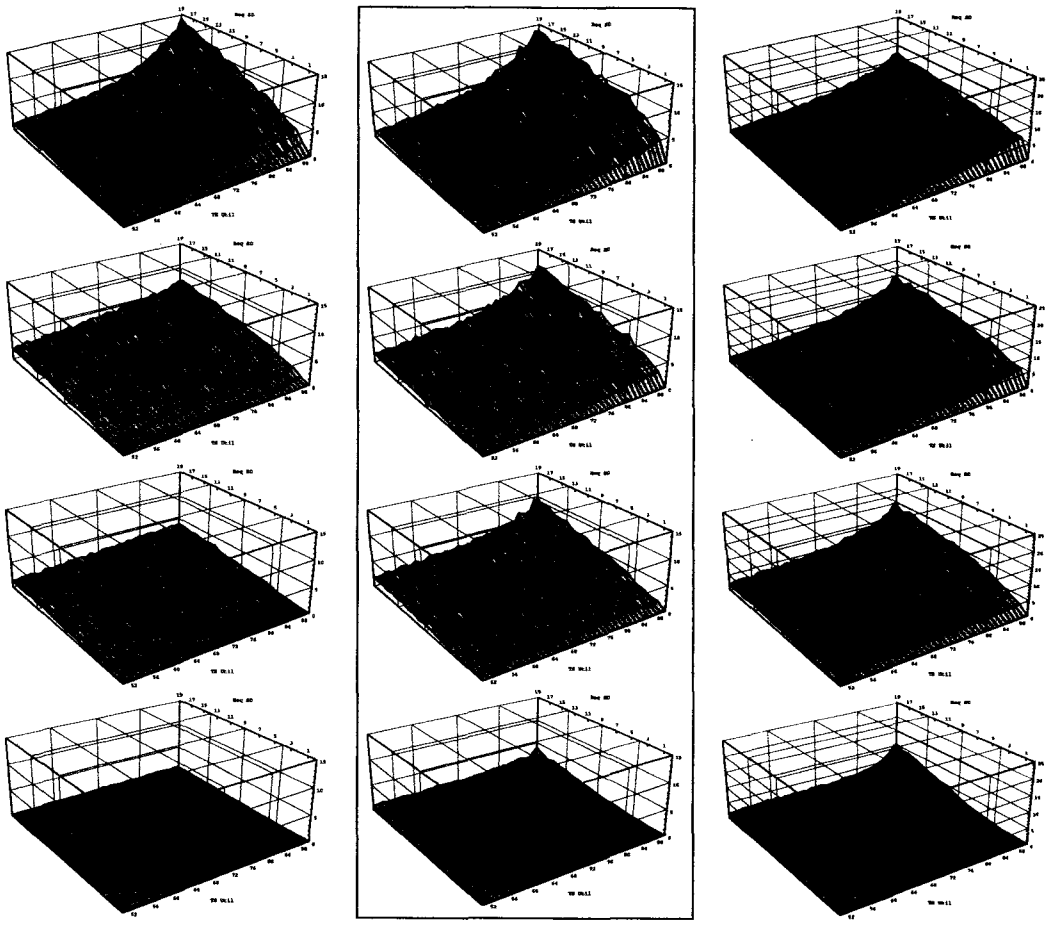
(b) Zero correlation between period and utilization.

(c) Positive correlation between period and utilization

Figure 5.12: INFLUENCE OF PERIOD/UTILIZATION CORRELATION ON PACKET LOSS: UNIFORM PERIOD DISTRIBUTION: *The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom. Figure 5.12(b) is identical to 5.9(a) for convenient comparison.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Negative correlation between period and utilization

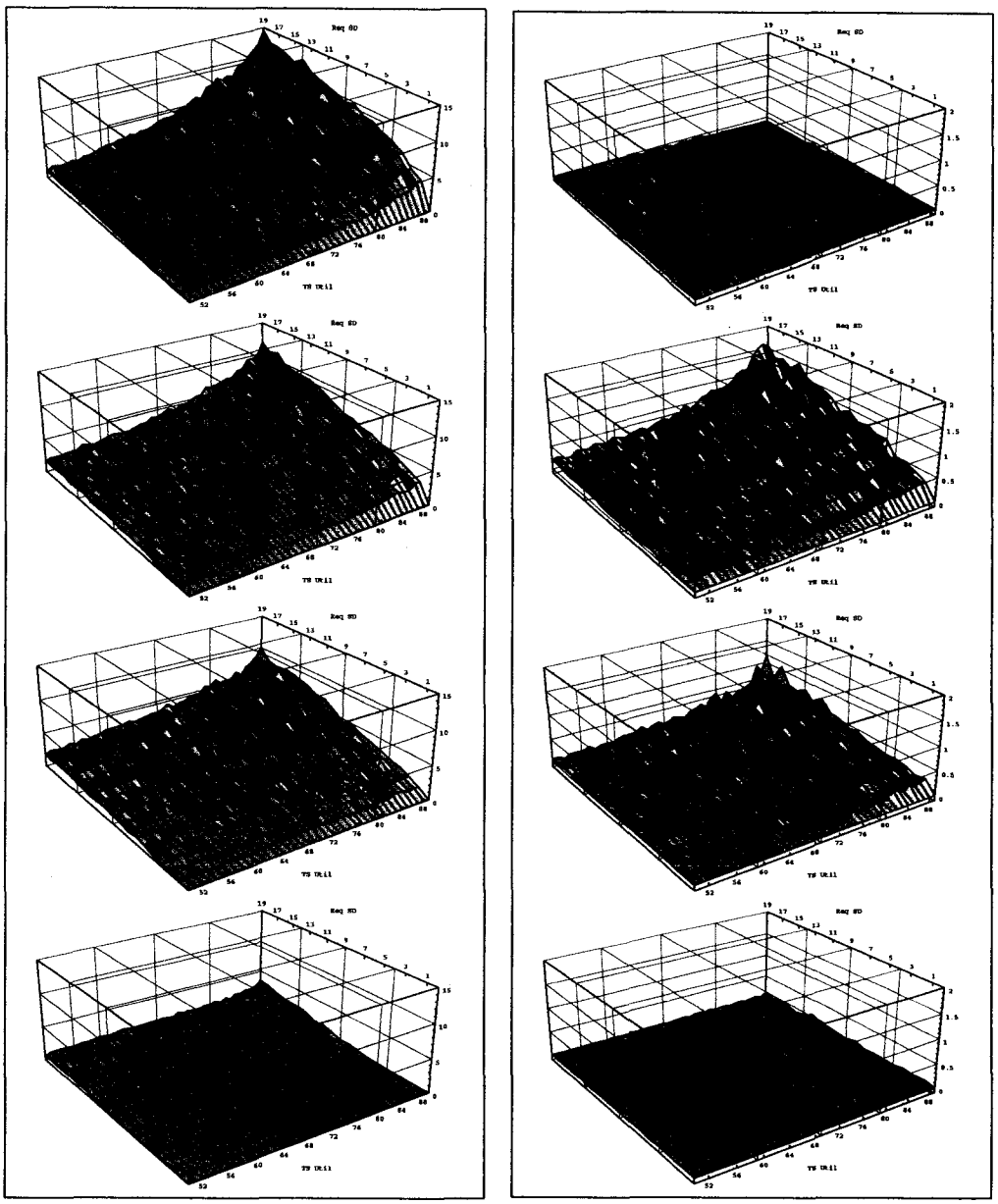
(b) Zero correlation between period and utilization.

(c) Positive correlation between period and utilization

Figure 5.13: INFLUENCE OF PERIOD/UTILIZATION CORRELATION ON PACKET LOSS: INDIRECT UNIFORM PERIOD DISTRIBUTION: *The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom. Figure 5.13(b) is similar to 5.9(a) for convenient comparison.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



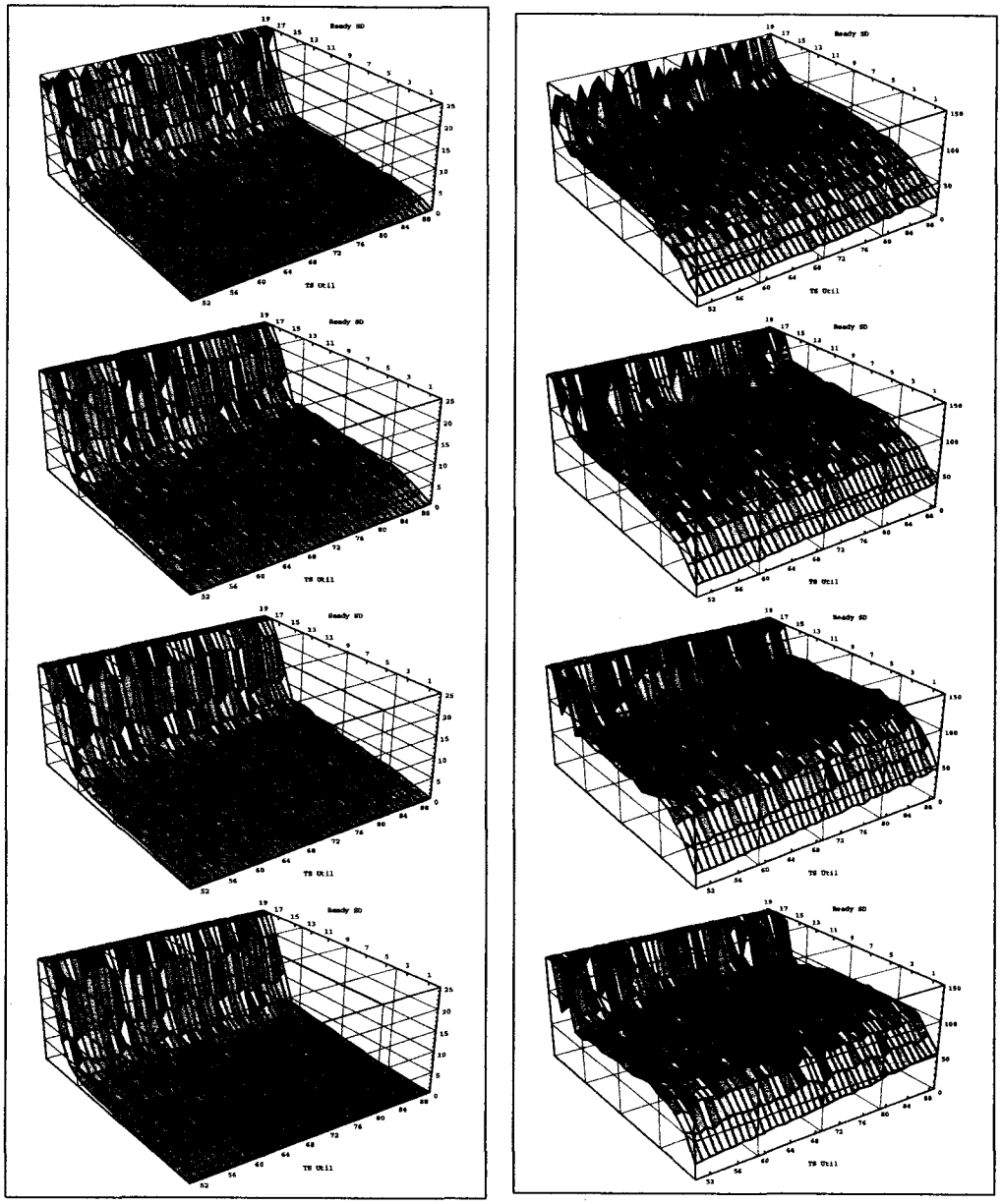
(a) Percentage of the messages dropped by RMS. The range of vertical axis is 0%-15%. This is a repeat of figure 5.9(a).

(b) The *additional* percentage of messages lost when idle time is evenly shared. The range of vertical axis is 0% to 2%.

Figure 5.14: INFLUENCE OF THE IDLE MANAGEMENT TIME SCHEME: *The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



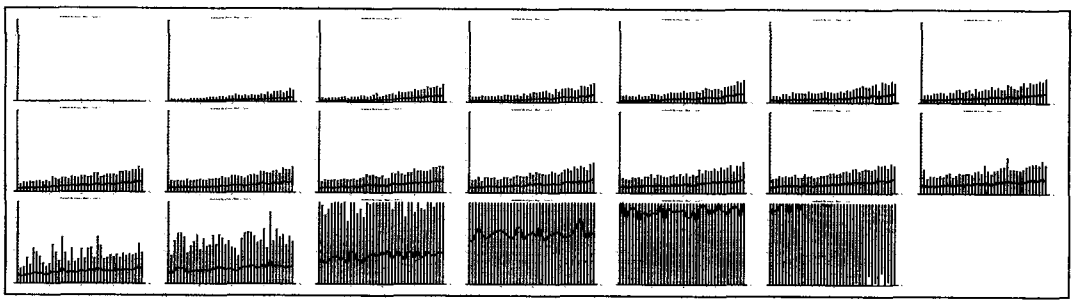
(a) Percentage of the messages dropped by RMS. The range of vertical axis is 0%-25%.

(b) Normalized message delivery times. The range of vertical axis is 0% to 150%.

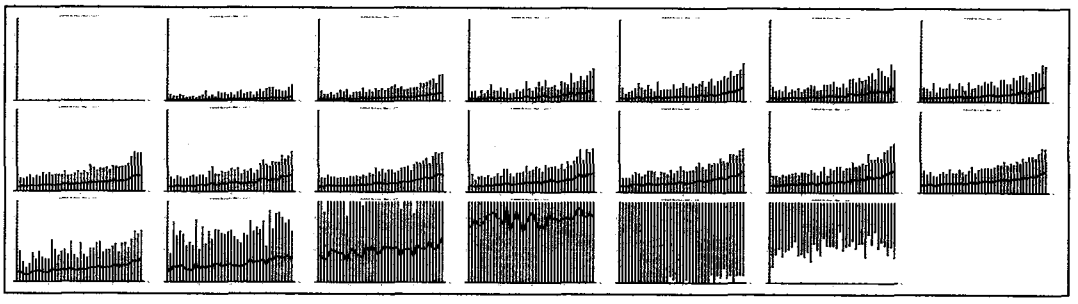
Figure 5.15: INFLUENCE OF MESSAGE READY TIME VARIATIONS ON RMS MESSAGE LOSS: *For each of the plots, utilization varies between 50%-90% and the StdDev for message ready time varies between 0%-20%. The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom.*



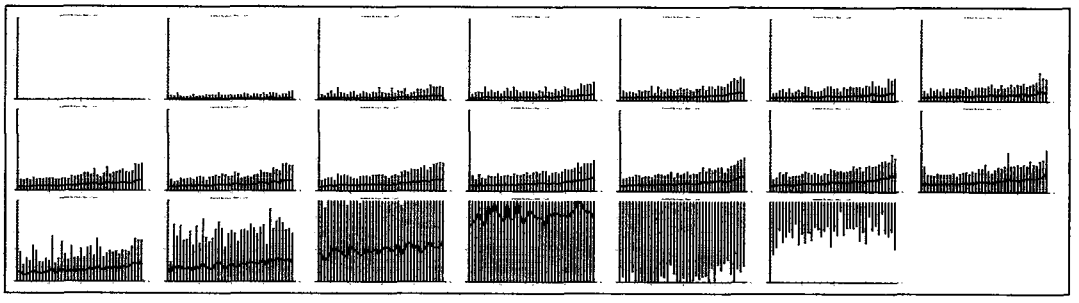
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



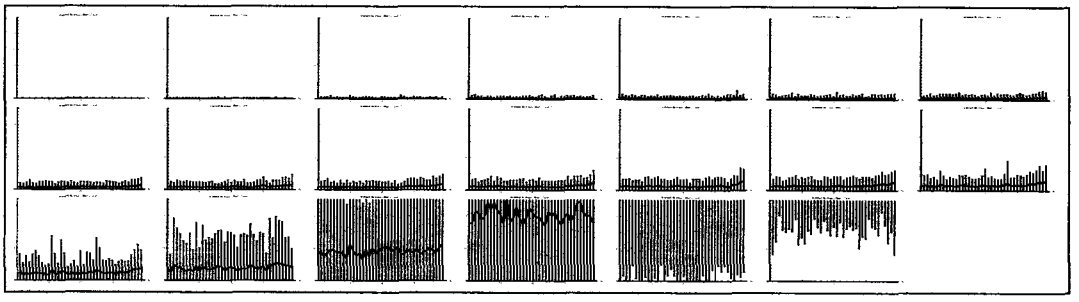
(a) Task $\tau_0$



(b) Task $\tau_1$



(c) Task $\tau_2$

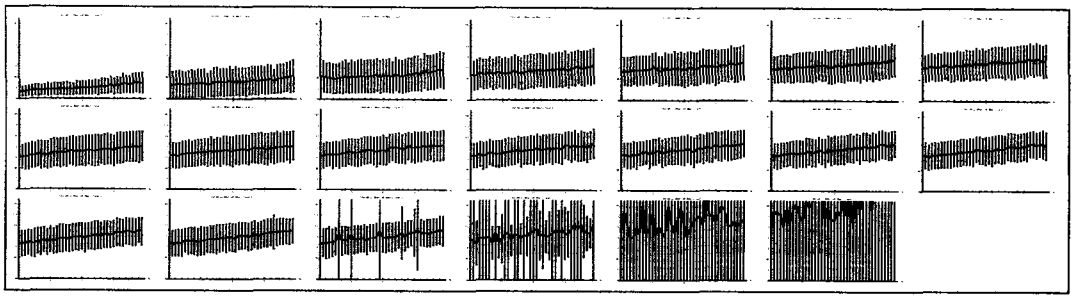


(d) Task $\tau_3$

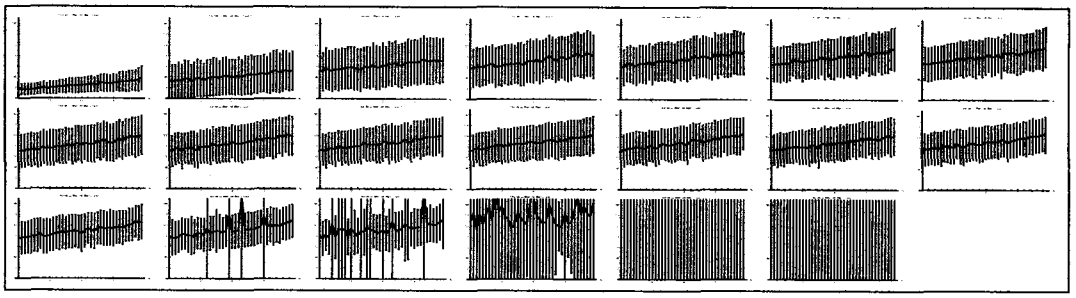
Figure 5.16: 2D Slices of surface plots in 5.15(A)



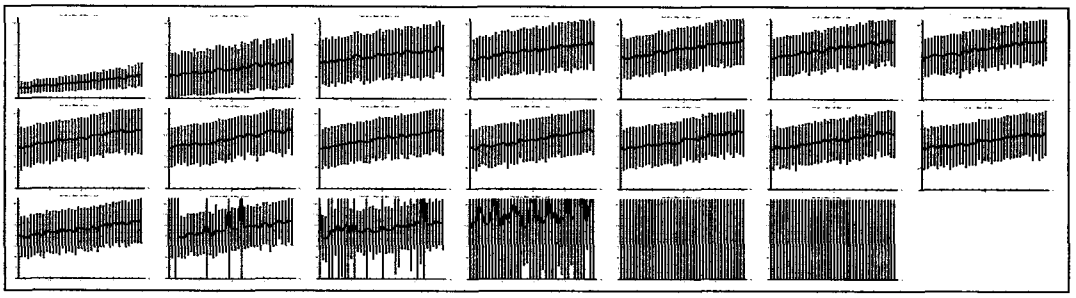
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



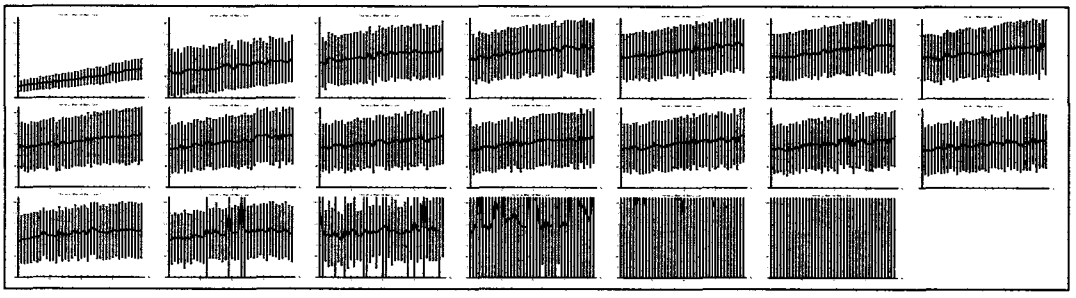
(a) Task $\tau_0$



(b) Task $\tau_1$



(c) Task $\tau_2$

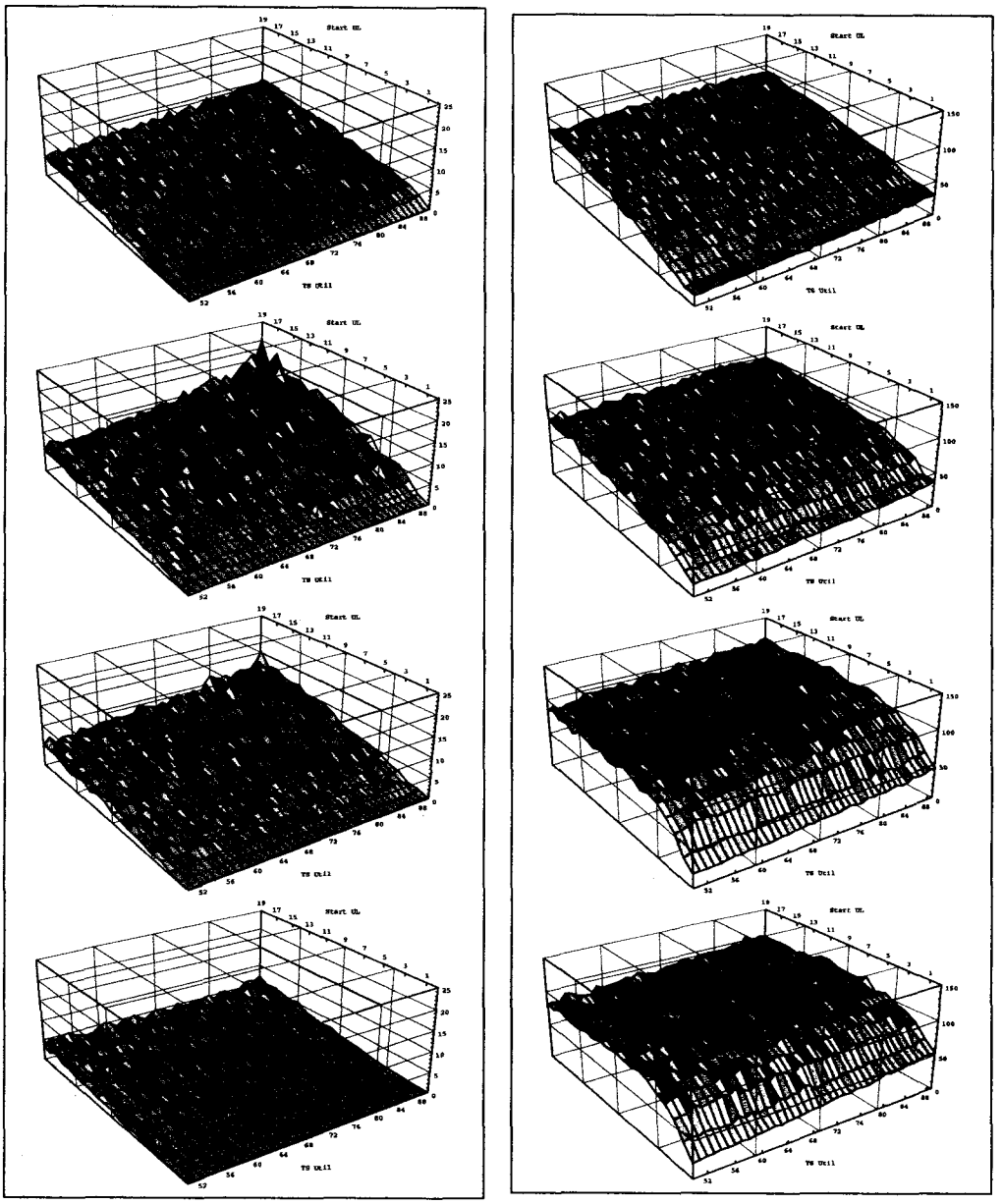


(d) Task $\tau_3$

Figure 5.17: 2D Slices of surface plots in 5.15(B)



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



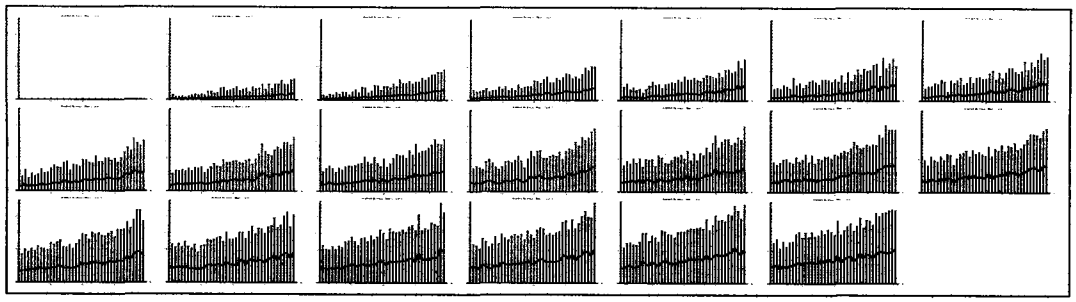
(a) Percentage of the messages dropped by RMS. The range of vertical axis is 0%-25%.

(b) Normalized message delivery times. The range of vertical axis is 0% to 150%.

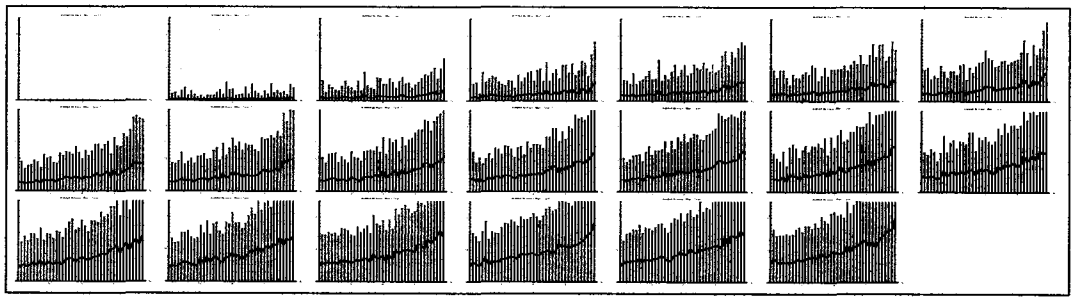
Figure 5.18: INFLUENCE OF STREAM START TIME VARIATIONS ON RMS MESSAGE LOSS: *For each of the plots, utilization varies between 50%-90% and the StdDev for stream start time varies between 0%-20%. The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom.*



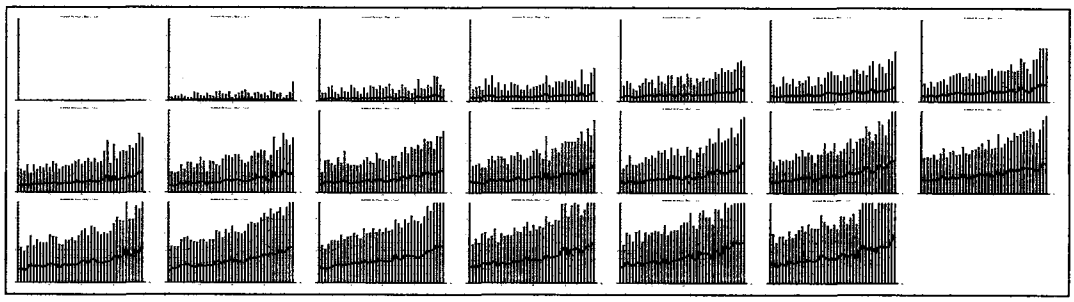
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



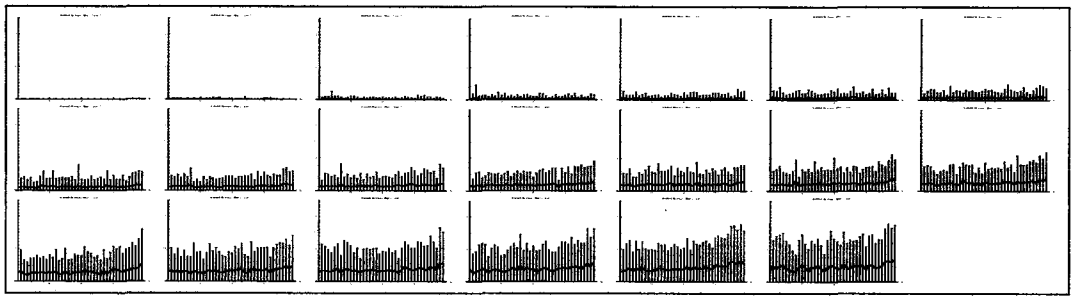
(a) Task $\tau_0$



(b) Task $\tau_1$



(c) Task $\tau_2$

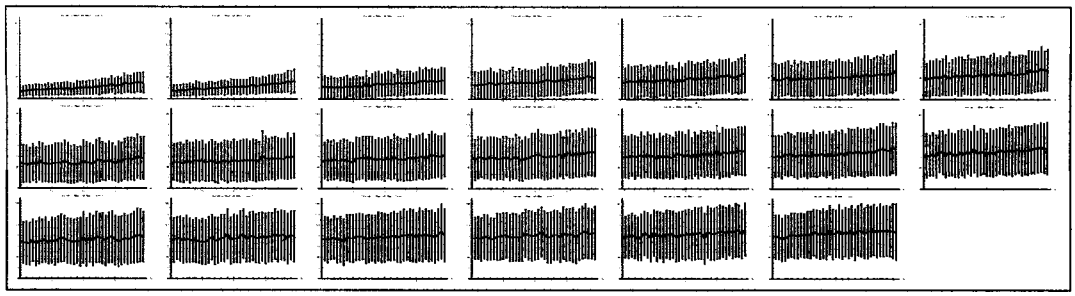


(d) Task $\tau_3$

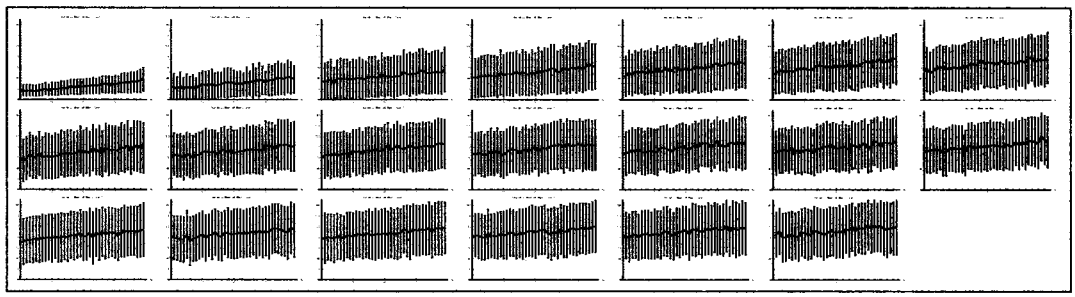
Figure 5.19: 2D Slices of surface plots in 5.18(A)



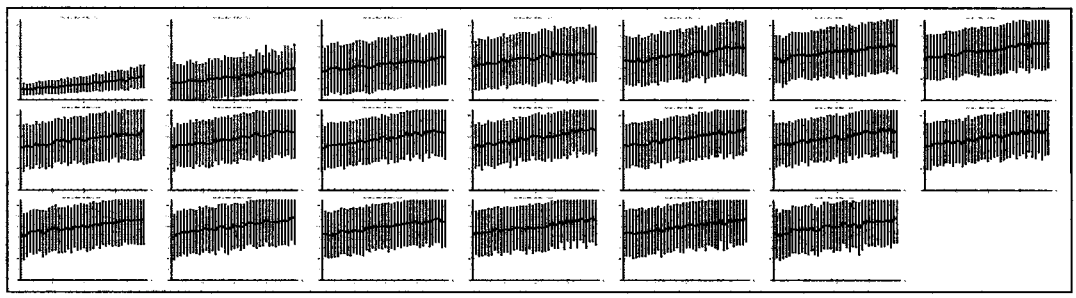
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



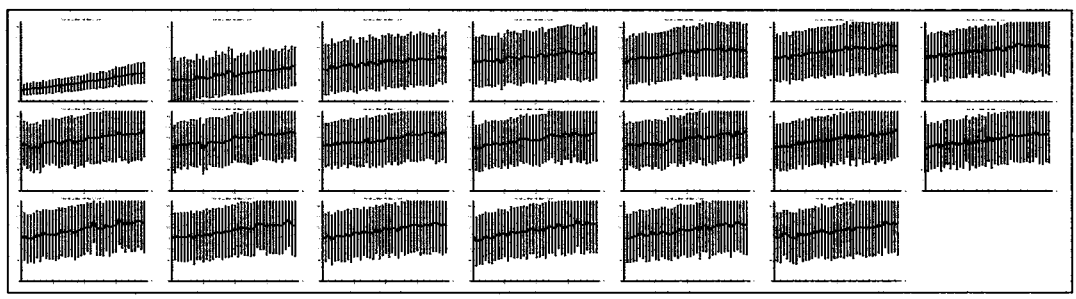
(a) Task $\tau_0$



(b) Task $\tau_1$



(c) Task $\tau_2$



(d) Task $\tau_3$

Figure 5.20: 2D Slices of surface plots in 5.18(B)



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# Chapter 6

# Distance Constrained Scheduler and Results

In chapter 5, we focused on the characteristics of the schedules created by the rate monotonic version of our network protocol; in this chapter, we focus on distance constrained version. We start with an outline of the operation of the scheduler (section 6.1). This is followed by a summary of the simulation results that investigate the handling of variations in message sizes, message ready times, and periodic stream start times (section 6.2). We conclude with a brief discussion of two algorithmic enhancements for better idle time management (section 6.3).

## 6.1 Operation Of The Distance Constrained Scheduler

As discussed in chapter 2, the basic idea behind DCS is a process called **specialization** which amounts to shortening the distance constraints for total alignment within task sets [HL89]. For the sample task set given in figure 6.1(a), two possible specializations are shown in figures 6.1(b) and 6.1(c); since the latter one leads to a smaller increase in the task set utilization, it is the better of the two. Our network scheduler is based on the specialization technique defined in [HL92] since it has been proven to lead to the smallest increase in



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

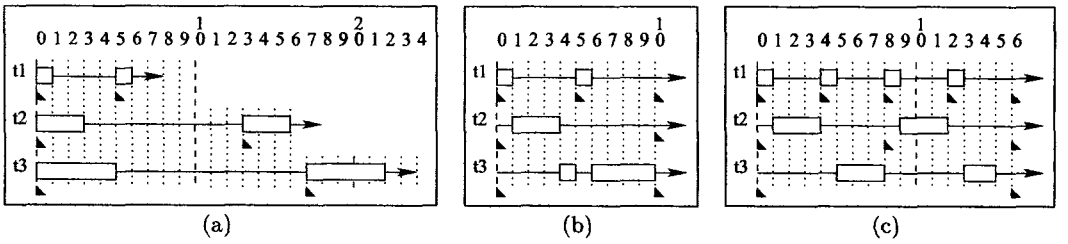



Figure 6.1: DISTANCE CONSTRAINED SCHEDULING: *As in RMS, we represent the ith distance constrained task using the notation* $\tau_i = (c, p)$ *where* $p$ *is the distance constraint and* $c$ *is the size of the resource request. The timeline representation of a sample task set* $\tau = \{(1,5),(3,13),(5,17)\}$ *is shown in part 6.1(a). In its original form,* $\tau$ *has a total utilization of* $\frac{1}{5} + \frac{3}{13} + \frac{5}{17} \approxeq 0.7249$. *When it is specialized with respect to* ***five*** *times the first* ***two*** *powers of two (i.e.* $5 \times \{2^0, 2^1\}$*), we get the schedule shown in part 6.1(b) with a new utilization of* $\frac{1}{5} + \frac{3}{10} + \frac{5}{10} = 1$. *When it is specialized with respect to* ***four*** *times the first* ***three*** *powers of two (i.e.* $4 \times \{2^0, 2^1, 2^2\}$*), we get the schedule shown in part 6.1(c) with a new utilization* $\frac{1}{4} + \frac{3}{8} + \frac{5}{16} = 0.9375$.

task set utilizations. As before, we factor in a host-specific "task switch overhead" during schedulability tests.

While RMS provides the resource as it is needed, DCS provides it more often than needed. This leads to a significant simplification in the design of the network protocol since the token visit times no longer need to be synchronized with message generation times; as long as the visits happen "often enough", every deadline is met. This also means that hosts no longer need to hold the token if they receive it for a SYNC event; if there is no message waiting to be transmitted, they simply relinquish their turn. In fact, in the distance constrained version of the scheduler, the SYNC flag of the token is not even used.

The representation of a schedule within the distance constrained scheduler is the same as before. Namely, events are arranged in groups of scheduling intervals. The definition of a scheduling interval, however, is slightly different. With RMS, each new job of each task starts a new scheduling interval. With DCS, we take advantage the regularity caused by the specialization process: only the jobs of the task with the shortest specialized constraint are used for establishing scheduling intervals. Figure 6.2 illustrates this process for the schedule shown in figure 6.1(c). One of the benefits of this arrangement is that the temporal



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

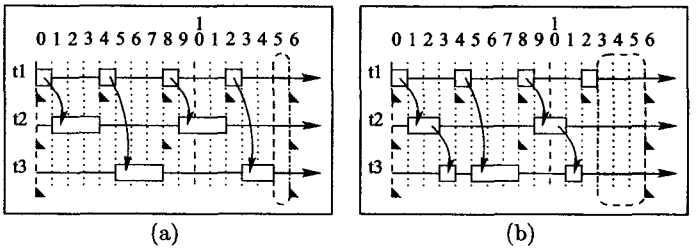



Figure 6.2: SCHEDULING INTERVALS WITH DISTANCE CONSTRAINED SCHEDULES: *The second of the two specializations we saw earlier leads to four scheduling intervals as seen in part 6.2(a). When the resource request of $\tau_t$ is reduced by one time unit, the number of scheduling intervals drop to three even though the number of events increase from eight to nine as seen in 6.2(b).*

duration of each scheduling interval (i.e. the sum of the event durations and any remaining idle interval) is the same.

We continue basing the operation of the scheduler around scheduling intervals for two reasons. First, they are used in idle time management. As the schedule is constructed, the amount of idle time available in an interval (which is the time saved from the events that last shorter than expected as well as the time slots allocated for terminated periodic streams) is recorded within the first event of that interval. The idle time is then used to increase the transmission windows of the member events as the schedule is used. The second reason has to do with slowing down the schedule in order to prevent the system from frantically passing the token back and forth when the hosts are underloaded. Since SYNC events are no longer used to inject delays, the scheduler implements an alternative "braking" mechanism. Specifically, at the end of each interval, the scheduler delays the transmission of the token by an interval equal to the shortest specialized distance constraint (i.e. the uniform length of any scheduling intervals) minus the actual durations of all the events in that interval. This initially sounds like another function being delegated to an already overloaded scheduler. However, the additional cost is practically zero. The scheduler already keeps track of event durations for idle time managements; a final subtraction to compute the required delay is insignificant. In addition, since the hosts no longer hold the token to wait for the generation



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of messages, the cost of measuring the duration of an event is much simpler because the scheduler no longer has to decide whether a token-holding delay has to be subtracted from the observed duration.

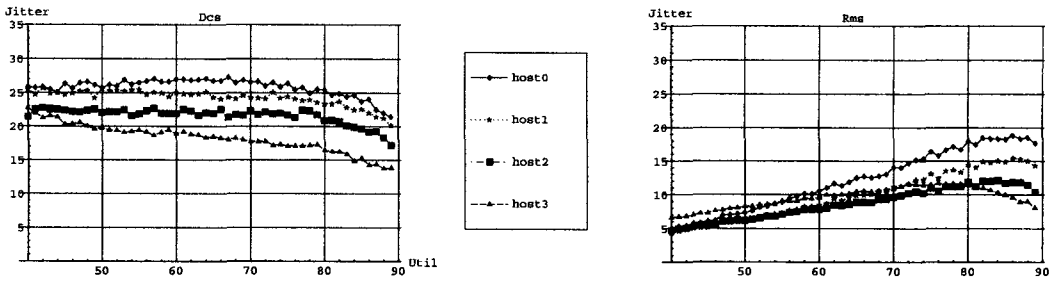
## 6.2 Performance of The Distance Constrained Scheduler

One of the performance consequences of providing the token more often than needed can be seen in figure 6.3. Part 6.3(a) is the jitter results of the distance constrained scheduler and part 6.3(b) is the corresponding result from the rate monotonic scheduler (which is a repeat of figure 5.7). We see that jitter has increased at every utilization level for every task in a way that is contrary to what is usually stated to be a strength of DCS. If the resource usage is not limited to discrete blocks and thus the resource requests can be scaled down by the specialization amount, then DCS does in fact produce schedules with zero jitter. In the context of networking, on the other hand, we deal with messages whose sizes cannot be scaled down simply because the resource is being granted more frequently than before. In this sense, we see that DCS is not the ultimate in terms of uniformity in interarrival gaps.

In figure 6.5(a), we show the effects of varying message sizes on the number of messages that are dropped by the scheduler; figure 6.5(b) is the corresponding results from the rate monotonic scheduler (repeat of figure 5.9(a)). As before, the two independent variables are (i) task set utilization, and (ii) the extent to which the message sizes differ from the mean values used during the construction of the schedule. The latter is quantified by setting one standard deviation of these changes as a percentage value of the mean message size (e.g. 10% on this dimension means that the standard deviation of the changes is set to 10% of the mean message size). The difference between the two algorithms is dramatic. Even though greater levels of variations are subjected to the distance constrained scheduler (standard deviation of the size variation goes up to 40% as opposed to the 20% limit for RMS), it consistently outperforms RMS. The main reason for this has to do with the clustering of events in the produced schedules. Rate monotonic schedules are far more specific to the



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.




(a) (b)

Figure 6.3: JITTER CHARACTERISTICS: *Part 6.3(a) are the jitter results of the distance constrained scheduler. At each utilization point on the x-axis, 250 task sets are randomly generated whose total utilizations are within half a percentage point of the target. A distance constrained schedule is then constructed for each task set and the jitter characteristics of the resulting events are computed. In order to keep the results independent of the underlying task sets, the jitter values of a task are normalized with the period of that task. The final numbers are therefore expressed as percentage values on the vertical axis. Finally, the average of the 250 normalized jitter values is plotted for each task at each target utilization. Part 6.3(b) are the associated results of the rate monotonic scheduler jitter results, repeated here for comparison.*

deadlines which means there are long stretches of scheduling intervals that have little or no idle time. Distance constrained schedules, on the other hand, have a more even distribution of idle intervals. In the previous set of plots, this had been the reason why DCS leads to greater levels of jitter; now we see the advantage of this difference, arguably making up for more than the jitter disadvantage.

In figure 6.6, we see the difference of the two idle time management techniques. The results in figure 6.6(a) (repeat of figure 6.5(a)) are based on distributing the idle time left in a scheduling interval on a first come first serve basis (pseudo code in figure 5.4(a)). The surface plot in figure 6.6(b) shows the extent of losing additional messages if we switch to sharing the idle time of scheduling intervals evenly among the member events of that interval (pseudo code in figure 5.4(b)). Since the former technique consistently works out better, we use it within all of the remaining simulations of this chapter.

The normalized transmission time of the messages under the distance constrained sched-



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

uler are plotted in figure 6.7(a) while those of the rate monotonic scheduler are in figure 6.7(b) (repeat of figure 5.9(b)). Unlike the message loss rate, we now see the rate monotonic scheduler outperform distance constrained scheduler. The main reason for this is that the particular specialization algorithm we used in the distance constrained scheduler provably causes the least amount of increased utilization. Consequently, the deadlines of the schedules it produces are met just in time. Using a different specialization algorithm which further reduces the distance constraints of the schedules (for those task sets that can afford this extra reduction) would likely have lower transmission durations. In order to view the variations of the results, figure 6.8(b) produces 2d slices through the surface plot of $\tau_0$ from figure 6.7(a) while 6.8(a) provides the 2d slices of figure 6.7(b). When task set utilizations are less than 80%, the variance for both algorithms is similar. Beyond this level, the variance increase in DCS is greater than that of the rate monotonic scheduler. Since what we see in figure 6.8 is representative of the general case, we did not provide the plots for the other tasks.

The surface plot in figure 6.9(a) represents the percentage of messages dropped when message ready times vary around the mean values used during the creation of the schedule. Figure 6.9(b) (which is a repeat of 5.15(b)) contains the corresponding results from the rate monotonic scheduler. It can be seen that the total collapse experienced by the rate monotonic scheduler does not happen with DCS. In addition, as the normalized standard deviation of the variations exceed 20%, the distance constrained scheduler experiences an improvement. The latter observation is due to the way the messages are generated in the simulation system. Specifically, if the randomly determined interarrival time of a pair of consecutive messages (at the source) is less than or equal to zero, then the second message does not materialize. The saved time can then be used to increase the transmission windows of the other events in the associated scheduling interval. Due to its greater resilience, the distance constrained scheduler performs acceptably until the normalized standard deviation of the ready time variations is large enough for this to occur. In the case of RMS, however, the scheduler collapses at lower levels. This difference can also be observed with the trans-



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

mission times of the messages shown in figures 6.10(a) and 6.10(b). Again, after around 20% in the message ready time variation dimension, the messages that do not get generated allow the distance constrained scheduler to transmit those message that do, reducing the average observed durations. The two dimensional slices of the transmission times for $\tau_0$ are also available in figure 6.11 (we do not have these slices for the message drop rate surface plots since those number are very close to zero in the case of DCS).

We finally compare the controlled variations in the start time of the periodic streams since almost all real-time periodic scheduling algorithms assume the tasks to start at time $t = 0$. Once again, DCS outperforms RMS due the latter's dependency on the occurrence of the events at the expected times. Figures 6.12 and 6.13 shows the message drop rates and normalized message delivery times, respectively. In this case, we do not see an improvement with the performance of DCS beyond the 20% mark since the random variation associated with each periodic stream applies equally to all the messages of that stream and thus no message cancellation occurs. However, further increases in this dimension does not make things worse (i.e. there is no uniform increase in this direction within the distance constrained plots of figures 6.12 and 6.13). The same is not true for RMS.

## 6.3 Additional Accommodations for Message Size Variations

Both of the scheduling algorithms discussed so far are capable of transmitting larger-than-average-size messages by making use of the idle time intervals of the schedule. As detailed earlier, the basic idea amounts to increasing the transmission windows of the events near these idle intervals to carry more than what had been reserved during schedule construction. Unfortunately, there are times when this scheme fails, even within close proximity of one or more idle gaps. Figure 6.4(a) shows one such case. If we assume that $\tau_2$'s second message ends up being three blocks instead of two, it cannot be transmitted since $\tau_2$ is not scheduled to get the token during the next scheduling interval which is where the extra time is available.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

If the scheduler could know the current state of all the hosts without any form of delay, then it could send the token where it is necessary whenever an idle time gap is big enough to squeeze in an unscheduled event. This, of course, is not possible in an ordinary network of computers. Conceivably, probabilistic schemes could be devised to statistically figure out which host is most likely to need extra transmission capacity at any given moment. There are, however, a few simple and non-probabilistic techniques that could be used during schedule construction to make a noticeable difference.

One of these techniques is based on distributing the idle gaps more evenly throughout the schedule. In figure 6.4(b), the transmissions of $\tau_2$ and $\tau_3$ are split into single-slot durations thereby creating single slot idle intervals in the first three scheduling intervals. Before the split, only the last message of $\tau_1$ has access to extra capacity; after the split, any three of nine messages can use the newly formed idle intervals.

Idle gap distribution can also be achieved by changing the way in which the schedule construction process proceeds. The creation of a distance constrained schedule starts by laying out the events for $\tau_1$ for the duration of an entire major cycle. The scheduler then iterates through the remaining tasks and schedules their events until the schedule is complete, or, until a deadline failure is detected in which case the task set is deemed unschedulable. Ordinarily, the events are added from the beginning of the major cycle to the end which clusters events at particular points (the worst being the first scheduling interval which contains an event for *every* task in the set). An alternative scheme is to schedule the even tasks from the beginning to the end and the odd ones in the reverse direction. Figure 6.4(c) shows the difference caused by this modification for the running sample task set. Even though the impact of this change is on $\tau_2$ only, we now have two idle gaps instead of one. Specifically, there is an idle gap after the second message of $\tau_2$ which means it can now transmit the problematic three slot message we mentioned earlier. In addition, in this this particular example at least, this modification comes at no cost: the number of events in figure 6.4(c) is exactly the same as the ones in 6.4(a).

Unfortunately, there is a cost to these types of schedule rearrangements. First, it can



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

be argued that although a greater number of events now have access to excess capacity, the schedule has lost its ability to transmit a message *much* larger than the anticipated size. Although it is difficult to precisely define the value of either scheme, the latter arrangement is likely to pay off more often as long as the messages generated by the periodic streams wander around an average without deviating too much (of course, if the messages size could be *radically* different than the parameters used during the construction of the schedule, there is not much that can be done with any know static scheduling scheme). A second and quantifiable cost is the increased overhead due to the increased number of events in a major cycle. The schedule in figure 6.4(b), for example, has three more events than the one in figure 6.4(a) and thus involves the transmission of three more tokens. If the total utilization is not high, then this may be acceptable; at higher utilizations, however, the increased overhead could make a schedulability difference.

Of the two idle time spreading mechanisms (i.e. break down of the messages into message fragments vs zigzagging during schedule construction), the latter is far better in terms of the extra overhead caused by the non-clustered nature of the schedules. For this reason, we used that technique in a set of simulations to get an estimate of the message drop reduction with DCS. Specifically, we repeated the simulations that produced the surface plots of figures 6.5(a) and 6.7(a) using zigzagging. Figure 6.14(a) plots the reduction in the rate of messages dropped. Since this surface is uniformly above zero level, we see that zigzagging has not cause any message lost. In fact, at almost 'every task set utilization' and 'message size variation' value combination, fewer messages have been lost. We also see that this gain is small with the highest precedence task $\tau_0$, increasing up to but excluding the lowest priority task. A similar pattern is observed in figure 6.14(b) which plots the reduction in the normalized message transmission time.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

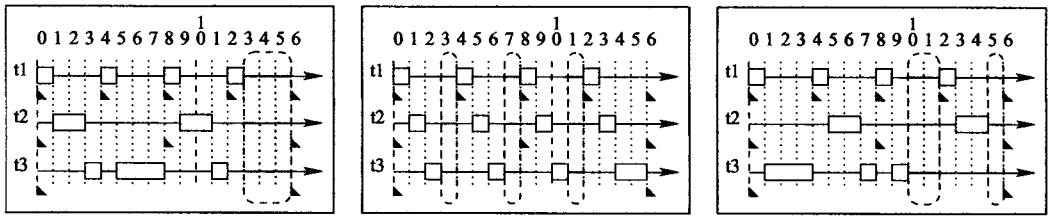


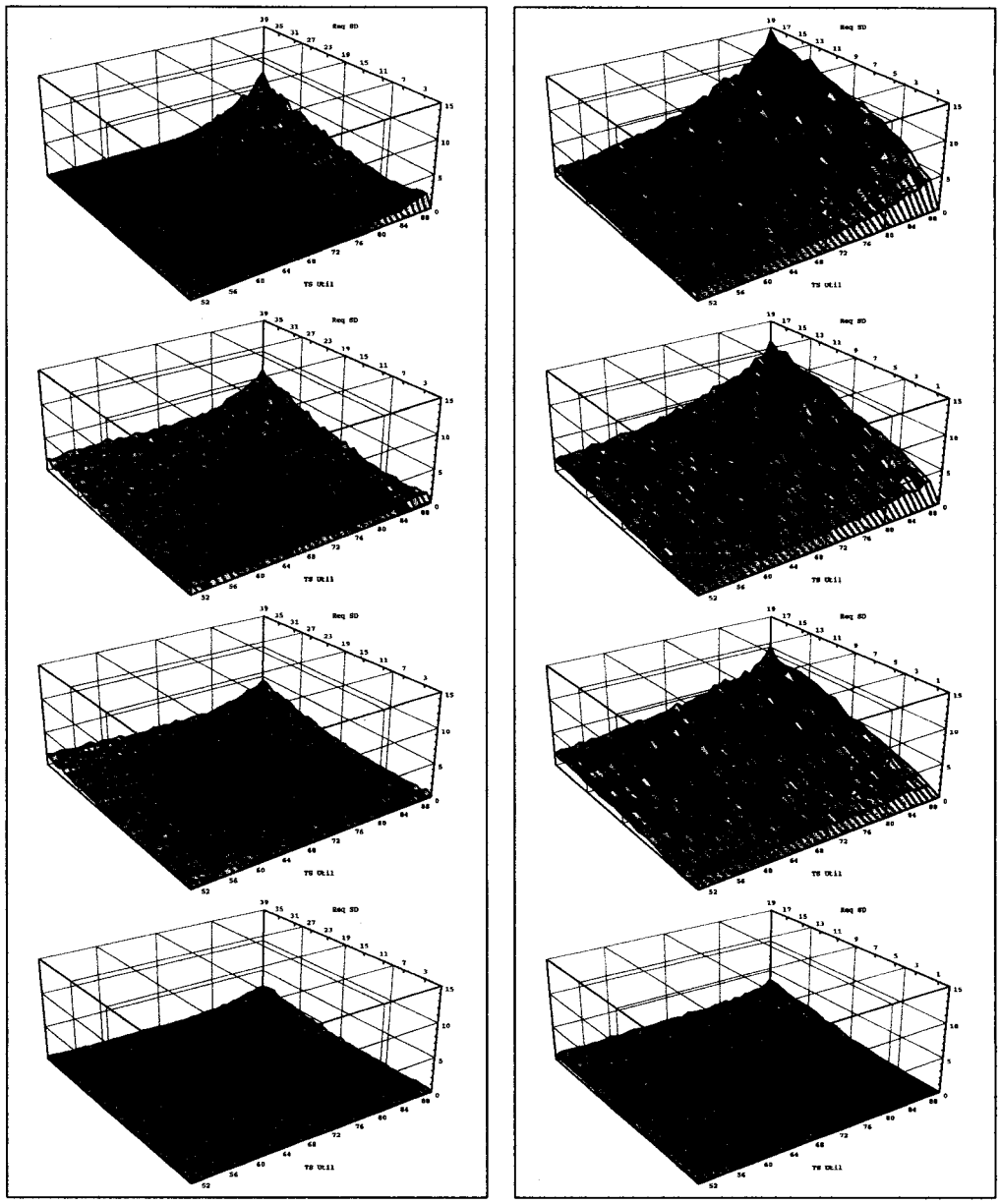

Figure 6.4: OPTIMIZATIONS ON DISTANCED CONSTRAINED SCHEDULES: *Part 6.4(a) is an ordinary distance constrained schedule. Part 6.4(b) shows the result of spreading the jobs of $\tau_2$ and $\tau_3$ over the scheduling intervals as evenly as possible. Part 6.4(c) shows the outcome of scheduling even numbered tasks from left to right and odd ones right to left; in this simple case, the only effect is on the events of $\tau_2$.*

## 6.4 Conclusion

Chapter 6 summarized the design and results of the real-time data-link layer network protocol based on DCS. A final overall comparison of the two schedulers is provided in section 7.1. Here, we conclude by noting that the performance difference between the two scheduling algorithms as far as handling variations in message ready times would have been even greater if we had used a slightly different model. In the experiments of chapter 5, the message ready times are off from the expected times by a randomly determined delta. An alternative model was to base the creation time of message $n$ on the creation time of message $n-1$. In the former scheme, the time deltas do not accumulate; in the latter, they do. If we were to rerun the simulations using the latter scheme, the relative performance of DCS over RMS would have increased due to its immunity of DCS to start time changes (which approximates the effect of the accumulation of ready time deltas).



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



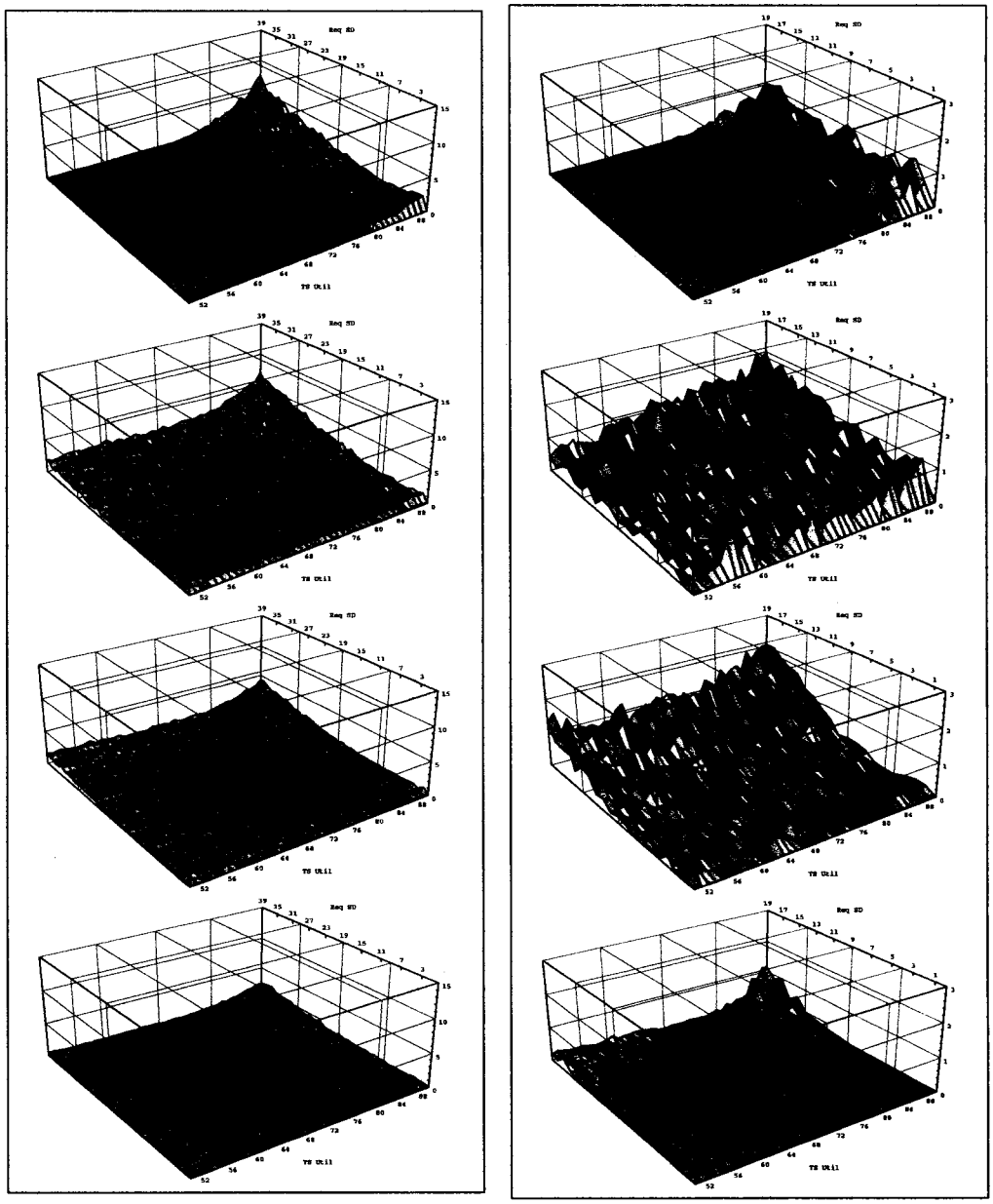
(a) **DCS:** StdDev for message size varies between 0%-40%. Range of vertical axis is 0%-15%.

(b) **RMS:** StdDev for message size varies between 0%-20%. Range of vertical axis is 0%-15%.

Figure 6.5: INFLUENCE OF MESSAGE SIZE VARIATIONS ON MESSAGE LOSS: *For each of the plots, task set utilization varies between 50%-90%. The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



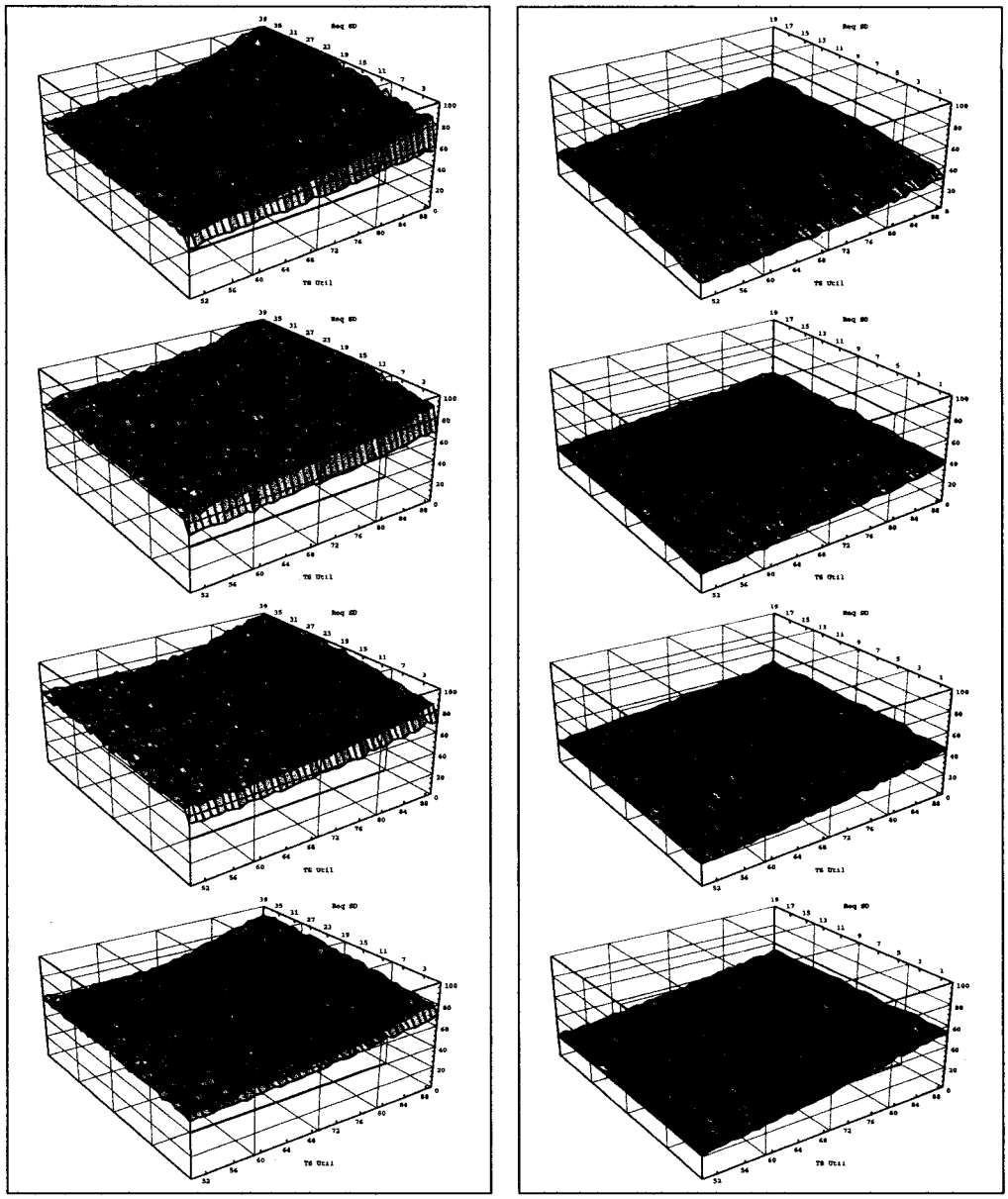
(a) Percentage of the messages dropped by DCS. The range of vertical axis is 0%-15%. This is a repeat of figure 6.5(a).

(b) The *additional* percentage of messages lost when idle time is evenly shared. The range of vertical axis is 0% to 3%.

Figure 6.6: INFLUENCE OF THE IDLE TIME MANAGEMENT SCHEME: *The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



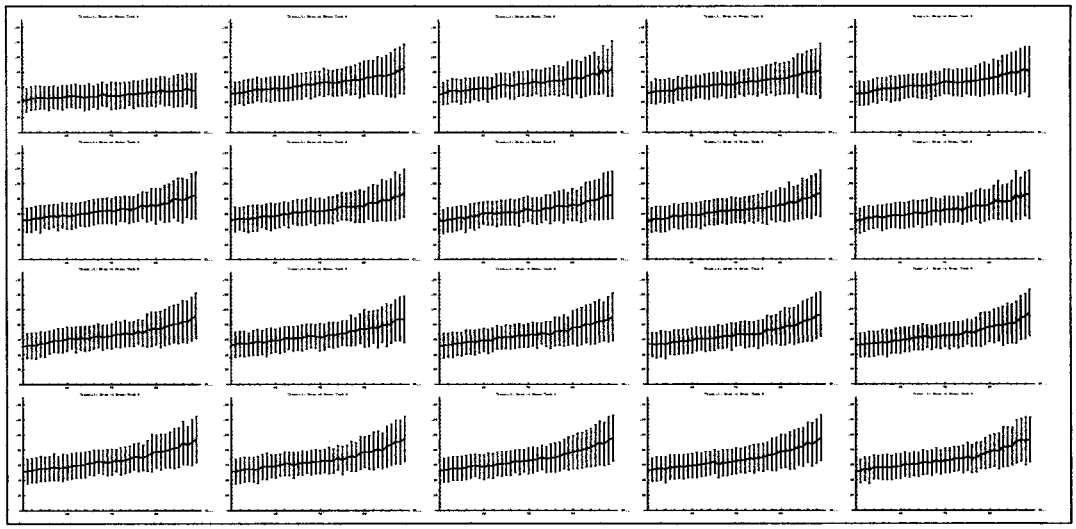
(a) **DCS:** StdDev for message size varies between 0%-40%. Range of vertical axis is 0%-100%.

(b) **RMS:** StdDev for message size varies between 0%-20%. Range of vertical axis is 0%-100%.

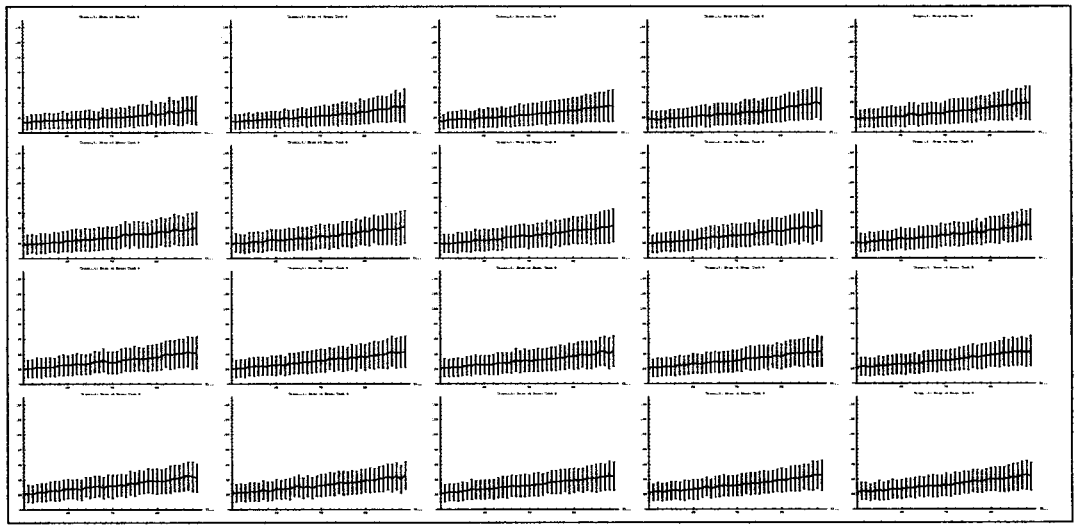
Figure 6.7: INFLUENCE OF MESSAGE SIZE VARIATIONS ON MESSAGE DELIVERY TIME: *For each of the plots, task set utilization varies between 50%-90%. The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) DCS results for $\tau_0$

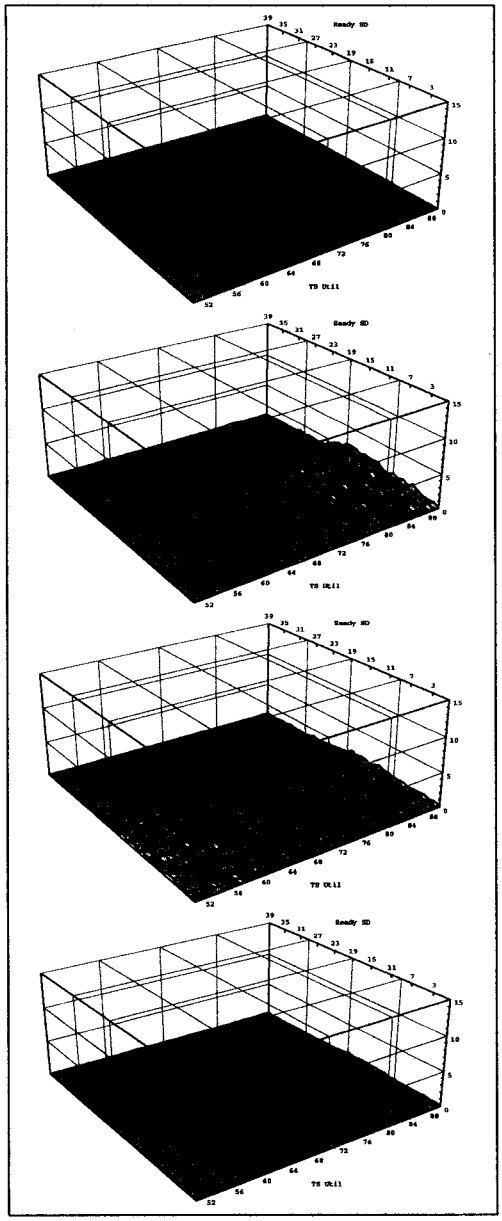


(b) RMS results for $\tau_0$

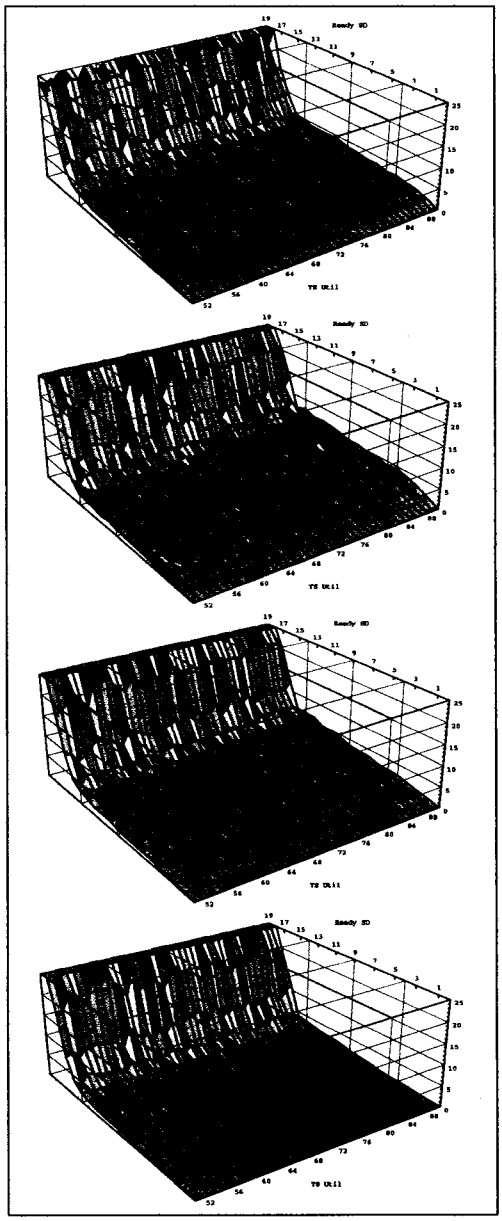
Figure 6.8: INFLUENCE OF MESSAGE SIZE VARIATIONS ON MESSAGE LOSS IN 2D SLICES FOR $\tau_0$: *Figure 6.8(a) shows the 2d slices for task $\tau_0$ from 6.7(a). Figure 6.8(b) (which is the same as $\tau_0$ in figure 5.11(a)) shows the case for RMS.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) **DCS:** StdDev for message size varies between 0%-40%. Range of vertical axis is 0%-15%.

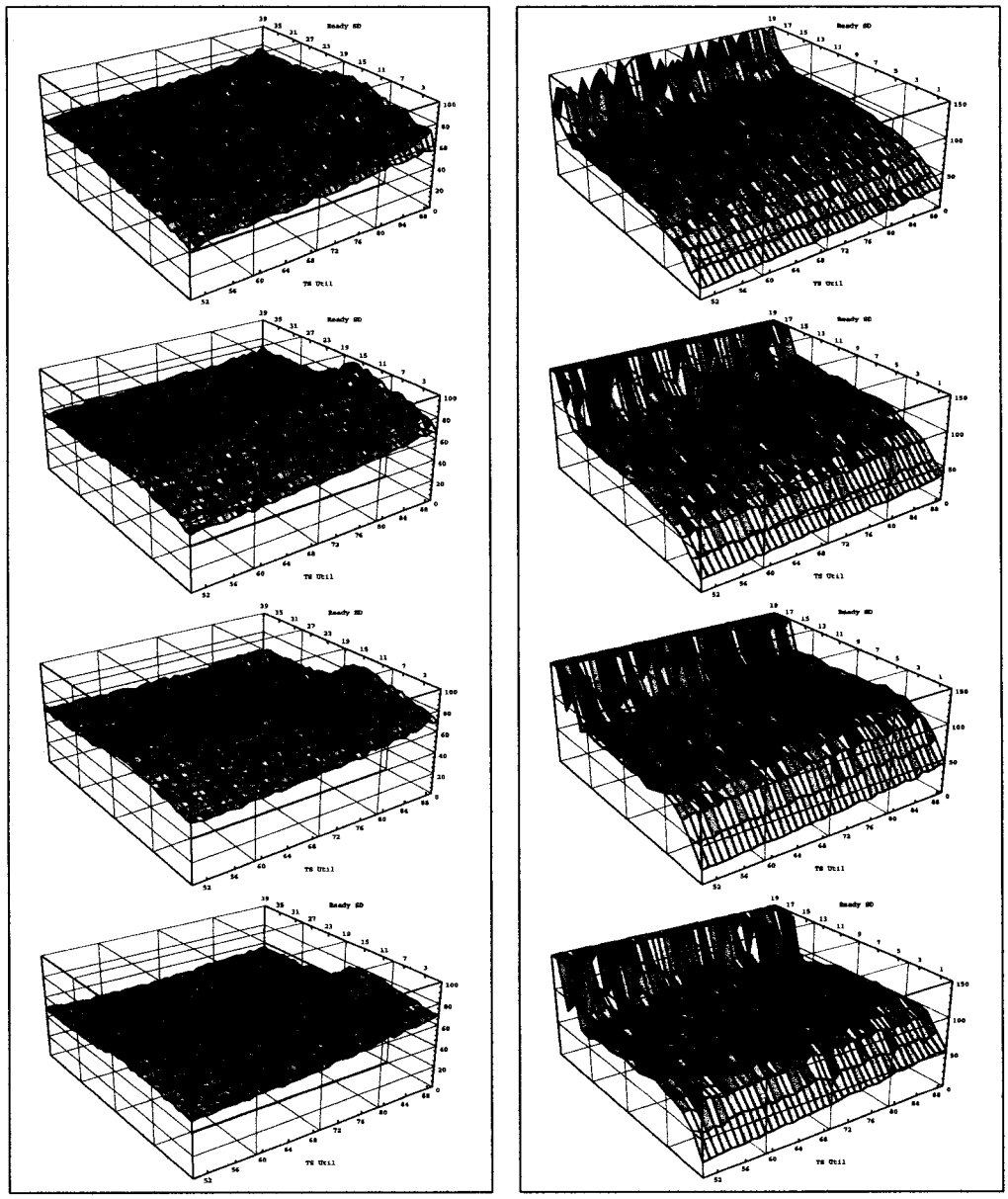


(b) **RMS:** StdDev for message size varies between 0%-20%. Range of vertical axis is 0%-25%.

Figure 6.9: INFLUENCE OF MESSAGE READY TIME VARIATIONS ON MESSAGE LOSS: *For each of the plots, task set utilization varies between 50%-90%. The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



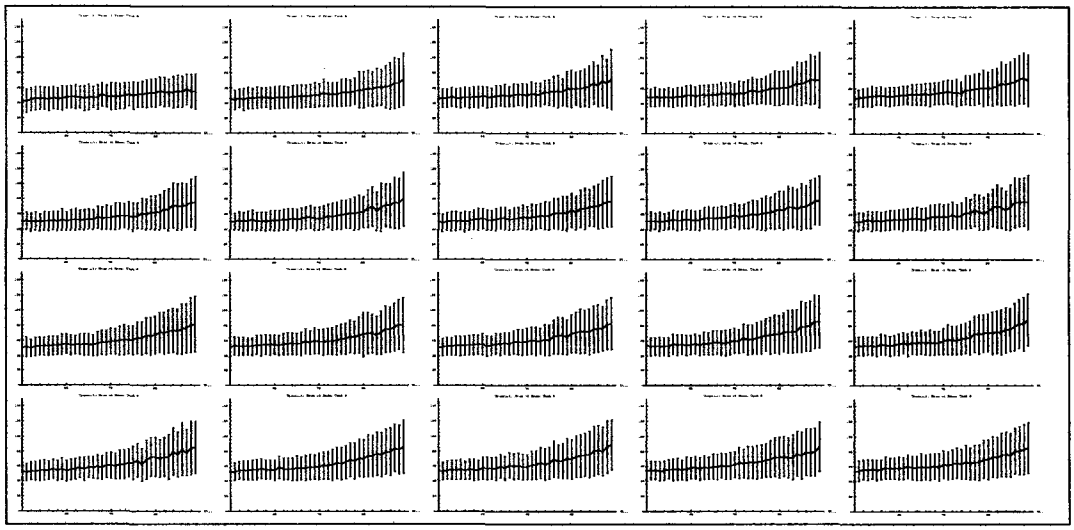
(a) **DCS:** StdDev for message size varies between 0%-40%. Range of vertical axis is 0%-100%.

(b) **RMS:** StdDev for message size varies between 0%-20%. Range of vertical axis is 0%-150%.

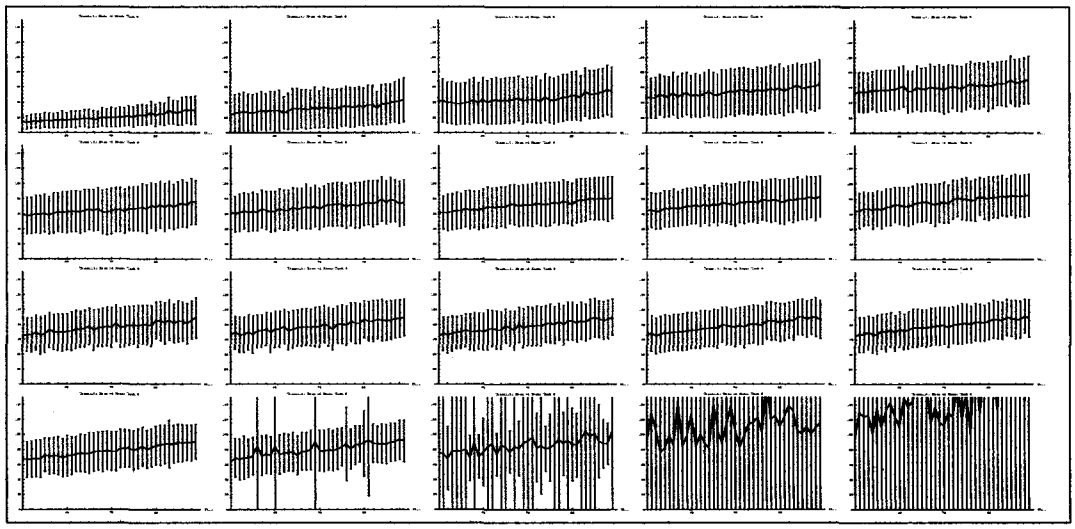
Figure 6.10: INFLUENCE OF MESSAGE READY TIME VARIATIONS ON MESSAGE DELIVERY TIMES: *For each of the plots, task set utilization varies between 50%-90%. The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Task $\tau_0$

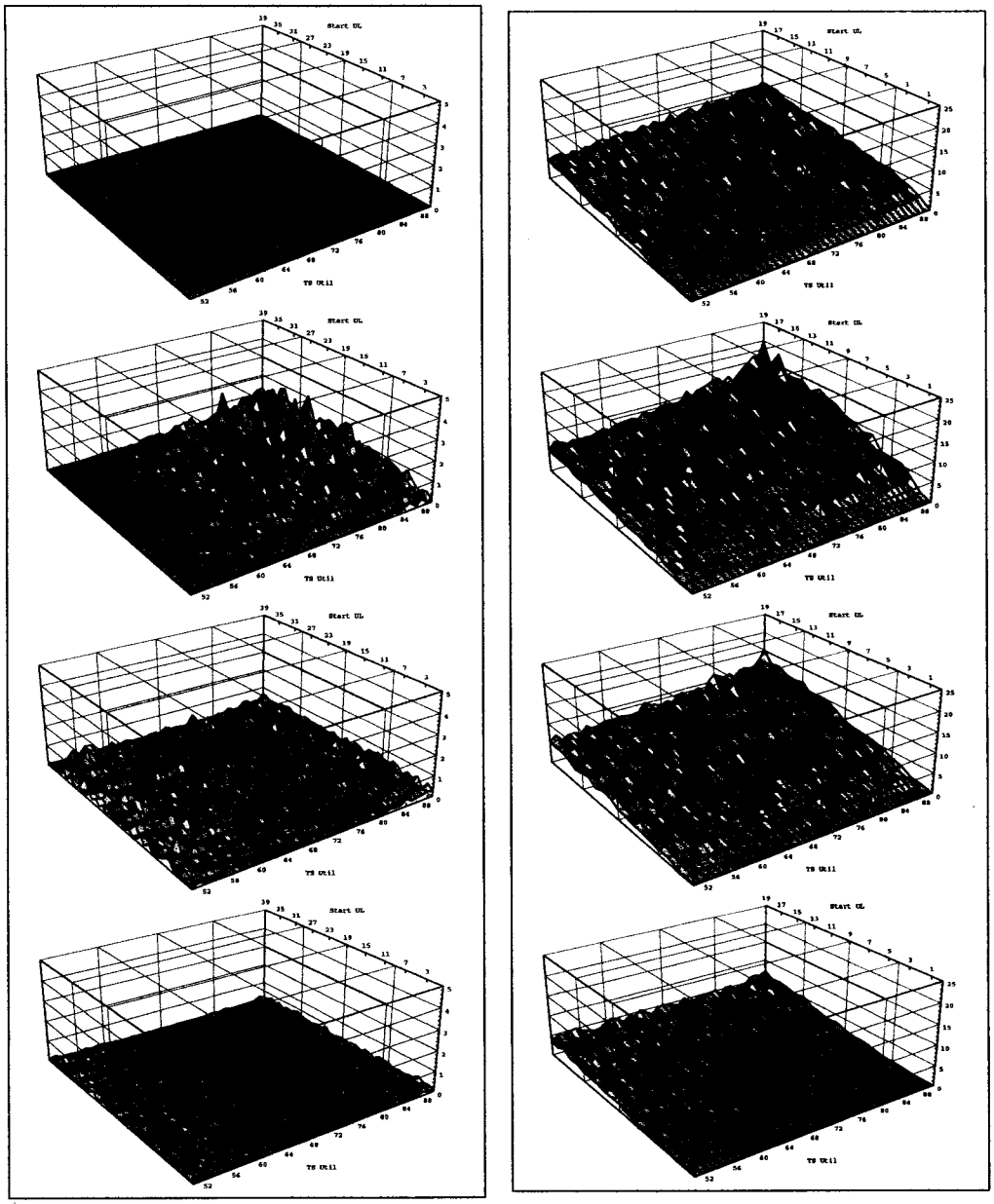


(b) Task $\tau_1$

Figure 6.11: INFLUENCE OF MESSAGE SIZE VARIATIONS ON MESSAGE LOSS IN 2D SLICES FOR $\tau_0$: *Figure 6.11(a) shows the 2d slices for task $\tau_0$ from 6.10(a). Figure 6.11(b) (which is the same as $\tau_0$ in figure 5.17(a)) shows the case for RMS.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



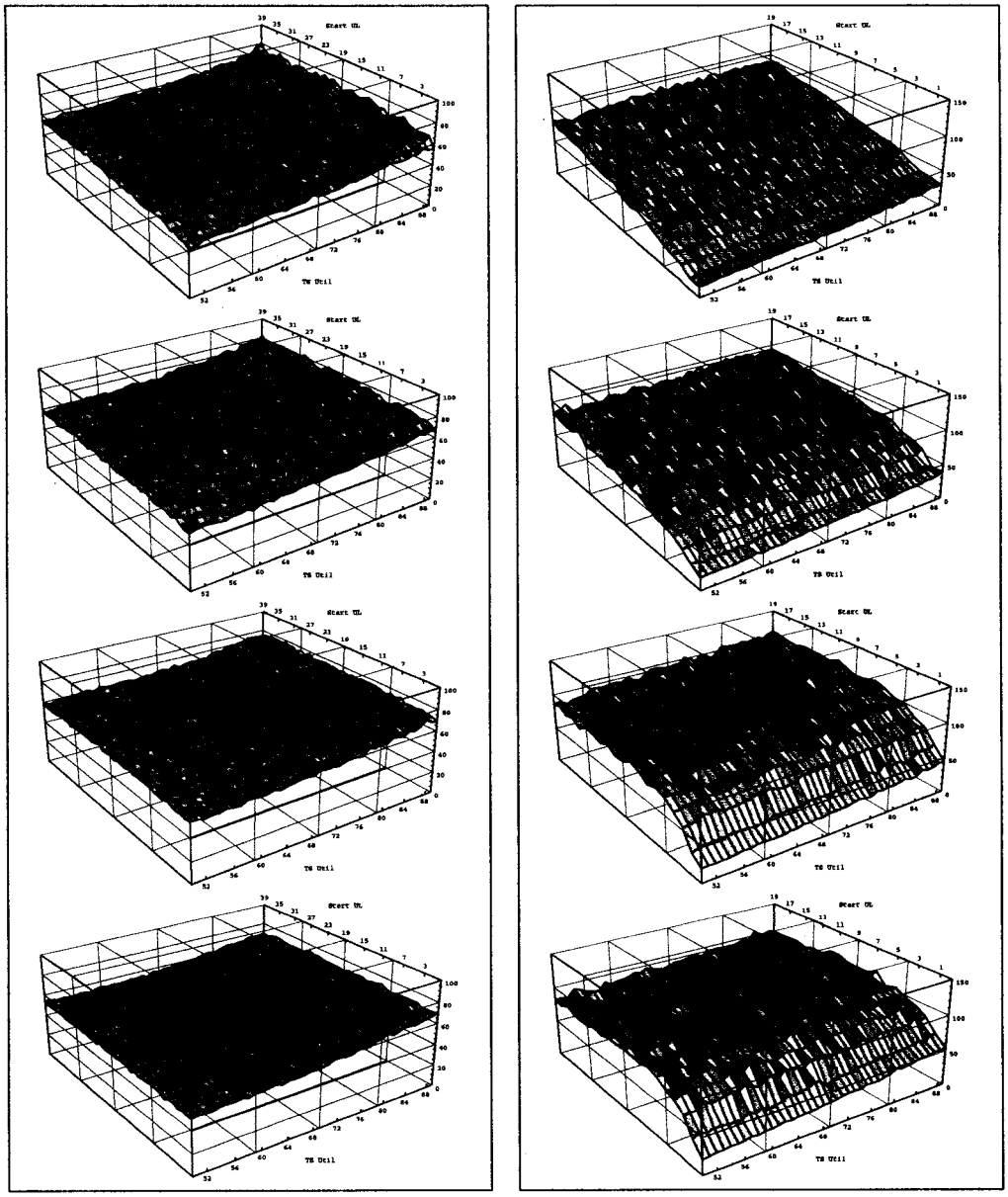
(a) **DCS:** StdDev for stream start varies between 0%-40%. Range of vertical axis is 0%-5%.

(b) **RMS:** StdDev for stream start varies between 0%-20%. Range of vertical axis is 0%-25%.

Figure 6.12: INFLUENCE OF PERIODIC STREAM START TIME VARIATIONS ON MESSAGE LOSS: *For each of the plots, task set utilization varies between 50%-90%. The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



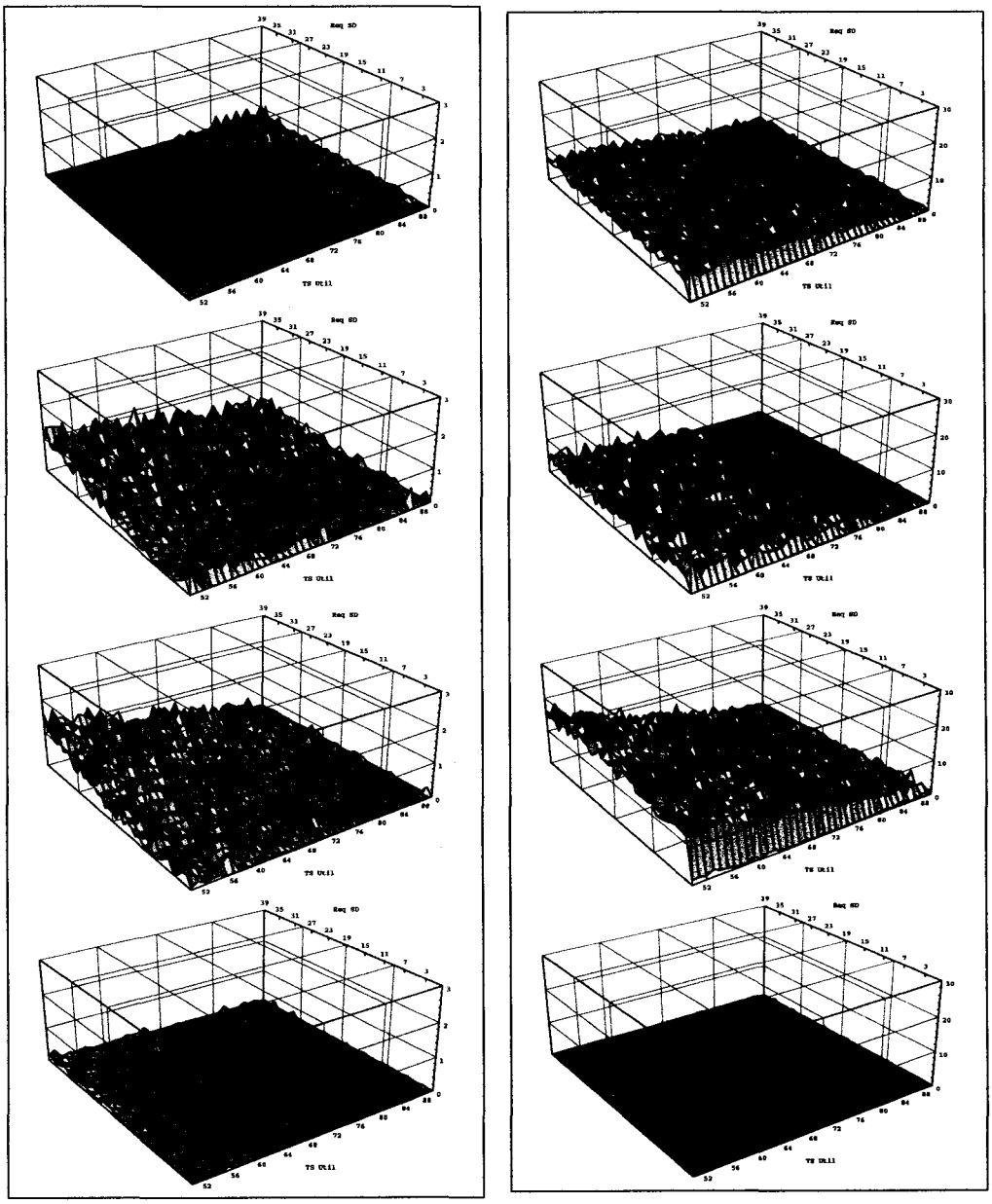
(a) **DCS:** StdDev for stream start varies between 0%-40%. Range of vertical axis is 0%-100%.

(b) **RMS:** StdDev for stream start varies between 0%-20%. Range of vertical axis is 0%-150%.

Figure 6.13: INFLUENCE OF PERIODIC STREAM START TIME VARIATIONS ON MESSAGE TRANSMISSION TIMES: *For each of the plots, task set utilization varies between 50%-90%. The tasks are ordered from highest priority ($\tau_0$) at the top to lowest priority ($\tau_3$) at the bottom.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Percentage improvement in message drop rates after activating zigzagging in DCS schedule creation

(b) Improvements in normalized message transmission time after activating zigzagging in DCS schedule creation

Figure 6.14: EFFECT OF ZIGZAGGING: *Zigzagging is a way in which clustered idle times can be distributed to a greater number of scheduling intervals with minimal increase in event handling overhead.*



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# Chapter 7

# Conclusion and Future Work

In this final chapter, we provide a summary of the results we presented in chapters 3, 5, and 6. We also discuss a number of ways in which our attempt to design a real-time data-link layer scheduler can be extended.

## 7.1 Summary of Results

Based on the following results and observations, we conclude that DCS provides an effective mechanism for implementing a periodic real-time data-link layer network protocol, specifically in relation to RMS:

- In chapter 4, we saw that RMS has better schedulability results than DCS when the task switch overhead is set to zero. However, this is not a realistic assumption for actual systems; in fact, network scheduling is associated with uncharacteristically high task switch overheads due to the use of the token packet which communicates the decisions of the scheduler. With non-zero task switch overheads, we see that the performance of DCS catches up with that of RMS. In addition, considerations of alternate orders within the same scheduling levels (which are caused by multiple tasks being mapped to the same specialized distance constraint) increase the schedulability rates of DCS even more.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- The simulations reported in chapter 6 have shown that DCS is far more flexible in absorbing variations in message size, message ready times, and periodic stream ready times. In addition, scheduling techniques such as zigzagging further increase this tolerance. Based on the results of chapter 5, on the other hand, we know that RMS is the better of the two algorithms when it comes to the minimization of message transmission time and jitter. These are, unfortunately, advantages that would disappear if multiple network segments were connected for establishing multi-hop periodic streams (this is a planned extension of our work, as stated in section 7.3). Since the schedulers of the different segments would work independently, messages would be subject to delays on bridge nodes as they transfer from one segment to the next. Since rate monotonic schedules are very sensitive to timing related changes, these delays would cause deadline failures and/or message loss.

- The specialization technique used within the experiments of chapter 6 was based on [HL92] which has been proven to have the best schedulability lower bound. Consequently, this technique is also known to minimize the amount by which the constraints are reduced. When schedulability is not effected, greater amounts of reduction may in fact provide greater idle capacity to accommodate message size variations. Therefore, we note that DCS has additional potential to deal with variations in periodic set descriptions. Allowing the network scheduler to pick different specialization schemes to be find the right balance between schedulability and the temporal characteristics of the produced schedules is a particular strength of DCS.

- As discussed in chapter 3, we prefer to construct and store schedules instead of computing them in a just-in-time manner for three reasons: (i) with non-zero task switch overhead values, the efficient schedulability tests of the two algorithm no longer work reliably; therefore, we now have to construct the schedule before concluding whether a task set is feasible or not; (ii) storing the schedule allows us to consider variations such as trying different priorities with individual specialization levels to boost



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

schedulability; this cannot be done with just-in-time scheduling; (iii) a data structure representation allows the scheduler to operate in $O(1)$ time per scheduling event. Due to the alignment effect of the specialization process, the temporal length of a distance constrained schedule is less than or equal to the longest distance constraint in the periodic stream set. The length of a rate monotonic schedule, on the other hand, is the least common multiple of the periods. This difference (which could easily be orders of magnitude) translates into substantial space savings for constructing and storage of schedules. This also simplifies the process of switching from one schedule to another. In a surveillance network, for example, the detection and tracking of new objects frequently trigger the creation of new schedules. We know from scheduling theory that unless the switch occurs at the completion of a major cycle, lower priority tasks run the risk of missing deadlines [TBW92]. Since distance constrained schedules are short, they minimize (or even eliminate) this possibility.

- We also consider DCS to be more accommodating for probabilistic considerations. We elaborate on this in section 7.2 along with a number of other mathematical issues.

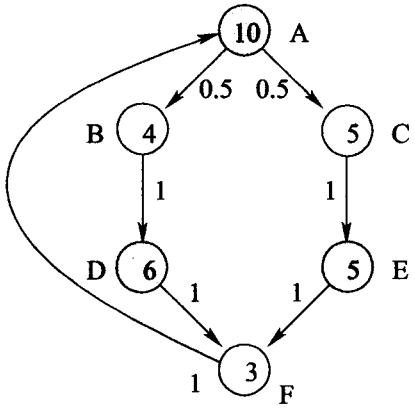
## 7.2 Mathematical Extensions

One of the exciting prospects at the beginning of this project was devising an algorithm to schedule task sets defined in probabilistic terms. As mentioned in chapter 1, our work in this regard was unfortunately not very fruitful. But the attempt deserves some space here since it will be a significant portion of our future work.

In classic scheduling theory, task sets are specified in terms of constants (e.g. constant resource requests, constant periods, etc). Actual systems, however, are not this simple. In fact, it could be argued that each parameter used to define a task, including the time of start, is determined by a probabilistic distribution. So an important question to address is how can the tight notion of real-time scheduling be merged with a probabilistic view and still offer temporally relevant results?



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.




(a)

| | A | B | C | D | E | F | $\mu$ |
|---|---|---|---|---|---|---|---|
| A | | 0.5 | 0.5 | | | | 10 |
| B | | | | 1 | | | 4 |
| C | | | | | 1 | | 5 |
| D | | | | | | 1 | 6 |
| E | | | | | | 1 | 5 |
| F | | | | | | | 3 |

(b)

Figure 7.1: A SAMPLE MARKOV PROCESS: *Part 7.1(a) is a Markov process of six states. Transition probabilities are written as edge labels and sojourn rates are written as node labels. The gray states represent the duration of network messages and while the white states represent the idle time between consecutive messages. Part 7.1(b) is the table view of the same system where the last column represents the sojourn rates.*

One option is to use Markov processes to model periodic streams. A Markov process is an elegant and simple type of stochastic system that is memoryless: a probabilistic description of what is about to happen with a Markov process is independent of its history. The elimination the memory reduces the generality of the model but also significantly simplifies analysis ([Cin75], [RMF96]).

The representation of a Markov process is very similar to that of a finite state machine. However, (i) states are labeled with mean sojourn times which, based on an exponential distribution, indicate the expected duration of stay at these states, and (ii) edges are labeled with transition probabilities as opposed to input symbols. In figure 7.1(a), a Markov model for a periodic stream that produces two sizes of messages is shown; figure 7.1(b) provides the equivalent table representation. Gray states correspond to the time it takes to transmit a message (and thus indirectly represent the size of the message) while white states correspond to the idle time between consecutively generated messages. The generation and transmission of two messages correspond to moving from state A back to state A. The ABDFA path



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

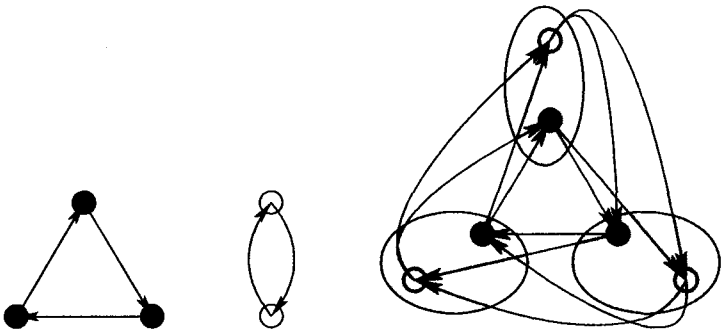



Figure 7.2: INTERSECTION OF TWO MARKOV PROCESSES: *The idea of the intersection of two finite state machines can be applied to Markov processes as well. Parts 7.2(a) and 7.2(b) of this figure show two simple Markov processes (the transition probabilities are intentionally not shown since our focus is the intersection process). Part 7.2(c) shows the intersection of these two processes where (i) the number of states is the product of the number of states of the two constituent processes, and (ii) the transitions accomodate all transition combinations for the two constituent processes.*

represents the generation of a medium size message (at D) following a large one (at A) while the ACEFA path represents the generation of a smaller message (at E) following a large one (at A). The mean idle state sojourn times are adjusted so that the expected duration from state A back to itself is the same regardless of the path taken.

The periodic stream modeled in figure 7.1 is simple but more complex configurations can be setup. For example, just by adjusting the transition probabilities out of state A, we can favor large-followed-by-medium size message pairs over large-followed-by-small pairs. By insertion of additional node pairs, we can attempt to model the variable bandwidth requirements of an MPEG compressed video stream.

If each periodic stream is represented as a Markov model, then the collective behavior of a set of periodic streams can also be represented in a Markovian manner with stochastic automata networks. The creation of a stochastic automata network is in many ways similar to creating the product (also known as "intersection") of two finite state machines. We illustrate the basic idea with an example. The product of the simple processes in figures



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

7.2(a) and 7.2(b) is shown in 7.2(c). The states in the product process is the Cartesian product of the states of the multiplied processes. The new transition probabilities, on the other hand, are computed with methods such as those provided in [LS04].

Markov processes are typically used to determine the steady state probabilities of the system being in any of the enumerated states. This is, by definition, time independent. The steady state probabilities of states A and B of figure 7.1 are shown with the horizontal lines in figures 7.3(a) and 7.3(b). As we would expect, the likelihood of being at state A is greater than that of state B since the mean sojourn time for the former is longer. Time dependent analysis (also known as transient analysis), on the other hand, focuses on where the system will be at a particular point in time and is specific to the start state. The likelihood of being at state A or B after starting at state A are shown with the curves in figures 7.3(a) and 7.3(b). The curve in 7.3(b) is especially revealing; we see that the likelihood of being at B peaks at around time $t = 6$ since we account for the time we must have spent at state A after the process had started. Before this time, it is likely that we were at state A; after this time, it is likely that we are at one of the downstream states.

Based on these techniques, it is conceivable to think that we can use time dependent analysis to determine the intervals of time during which a set of periodic streams are likely to overload the system and cause deadline failures. We can start with the Markov models of each periodic stream and compute their product. As outlined above, the resulting state space will be the Cartesian product of the state spaces of the individual processes. In addition, each of these new states will represent a different combination of idle vs message transmission intervals, some having a greater number of message transmission intervals than others. We can then use time dependent analysis to compute when we are likely to hits these events. We can also phase the start times of the constituent processes in an effort to eliminate the number and duration of such problematic states.

The process sketched above is computationally intense. But the bigger problem turns out to be the lack of precision we are forced to work with as a result of using the exponential distribution in determining the sojourn times. If any other type of distribution is



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

used, then the system is no longer memoryless and can thus no longer be analyzed with the techniques associated with Markov models and stochastic automata networks. If the exponential distribution is used, we are forced to work with great levels of variance, quickly degenerating time dependent predictions into steady state figures (as what happened at around time $t = 16$ in figure 7.3(b)).

Our attempt to use phase type distributions and other means to reduce the variance while holding on to memorylessness did not produce notable results under the time constraints of this project. However, work along these lines still seems to be a logical pursuit for the long run. Scheduling theory is very incomplete in terms of quantifiably soft real-time scheduling. Compared to what we know in hard real-time scheduling, our knowledge in algorithms with specific and probabilistic deadline guarantees needs much progress.

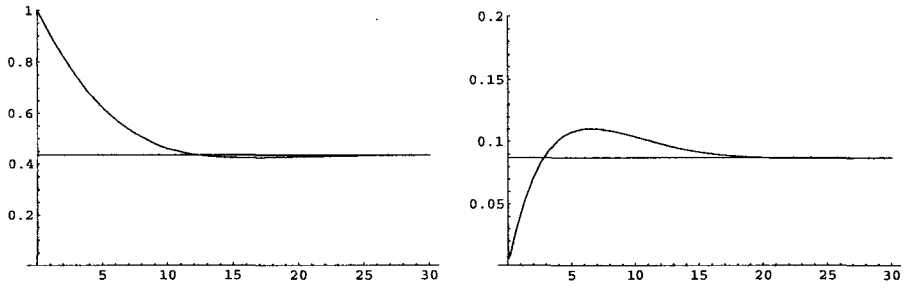
In the short term, however, we plan to apply probabilistic considerations to the two schedulers we discussed in chapter 5 and 6:

- DISTANCE CONSTRAINED SCHEDULING: The specialization process offers the shared resource to the competing tasks more often than needed. This means that, depending on the way in which specialization works out for a specific task set, the resource occasionally goes unused. This is illustrated in figure 7.4 where the timelines of two periodic tasks are drawn out. Task $\tau_1$ with period/request values of $5/1$ is specialized to a distance constraint of 4; task $\tau_1$, with a period/request values of $9/2$, is specialized to a distance constraint of 8. We can see that during its second job, $t_1$ cannot utilize the resource at the instant it gets it since its second request is yet to occur. Based on the periods and the distance constraints computed after specialization, we should be able to compute the probability of this happening to each task. We could then use a convolution of these distributions to determine a model of the idle time availability in each scheduling interval.

- RATE MONOTONIC SCHEDULING: The concept of a scheduling interval for both of the schedulers we developed is based on the idea of blocks of contiguous requests, each



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.




(a) (b)

Figure 7.3: STEADY STATE AND TIME DEPENDENT TRANSITION PROBABILITIES: *The horizontal line in parts 7.3(a) represents the steady state probability of being in state A while the curve represents the time dependent probability of being at A when already having started at A. Similarly, the horizontal line in parts 7.3(b) represents the steady state probability of being in state B while the curve represents the time dependent probability of being at B when already having started at A.*

starting with a synchronization event. In the case of RMS, the original task parameters are not modified (i.e. there is no specialization) and thus the lengths of the busy intervals are irregular (the extent of this irregularity depends on the particular periods and requests found in the task set). Using time dependent analysis, it is conceivable to associate a probabilistic term with each busy interval, stating the likelihood of finding additional idle time even the interval appears full based on expected values. This addition would create a better idle time management mechanism for RMS and improve its performance.

## 7.3 Implementation Related Extensions

As mentioned in chapter 1, our work focused on the single segment version of the network scheduling problem. We now plan to extend our efforts to wider contexts. Specifically, we plan to apply real-time networking to Mobile Adhoc NETworks (MANETs) which, by definition, consist of multiple segments. Below is a rough list of steps we plan to take along these lines:



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

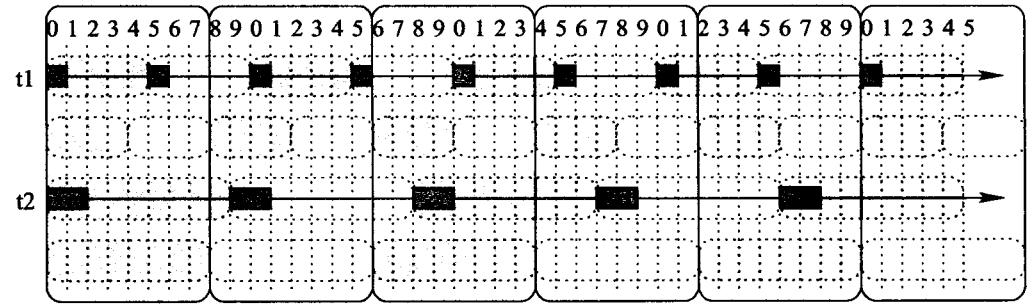



Figure 7.4: PROBABILISTIC CONSIDERATIONS WITH DCS: *This diagram shows the original and the specialized forms of two tasks: $\tau_1$ with a period/request of 5/1 and $\tau_2$ with a period/request of 9/2. In the first (i.e. topmost) timeline, the alternating gray/white boxes show the job boundaries for $\tau_1$. In the second timeline, the alternating gray/white boxes show the intervals job intervals for $\tau_1$ after it has been specialized by 4. The third and the fourth timelines show the same things for task $\tau_2$ which is specialized by 8. When we consider the times in which the shared resource is needed by these two tasks and the job boundaries of their specialized forms, we see that some of these jobs will go idle simply because the resource is being offered more often than needed (which is one of the key characteristics of distance constrained scheduling).*

- We first need to port the protocol to a new simulation system that can be conveniently used for all of what we outline in this section. The existing work was implemented largely as stand alone as well as OPNET simulations and is not as clean as one would wish to use as a base for future work. Of the various event and process oriented simulation systems that exist today (such as ns2, Simula, Parsec/GloMoSim, OPNET, etc), we plan to use JiST which came out of Cornell University in 2004. Its main strengths are its scalability as well as its clean Java interface for coding. The former quality is important due to our planned work on MANETs; with conventional monolithic systems, conducting experiments beyond a few nodes is usually very time consuming. The latter quality is important since we can use Java and (more importantly) its API for more compact and clean implementations which makes it possible to accommodate undergraduates in this work.

- Once the port to JiST is complete, we plan to integrate our work with the IEEE 802.11 family of protocols. Specifically, we plan to make use of the Point Coordination Function (PCF) hook which, as stated in [LGW03], "offers the capability to provide



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

connection-oriented, contention-free services by enabling polled stations to transmit without contenting for the channel; the method by which polling tables are maintained and the polling sequence is determined is left to the implementor". We may also choose to extend the protocols at the higher levels of the stack so that the use or cancellation of the real-time mode can be under the control of the application or transport layer entities.

- By definition, a MANET consists of multiple segments since mobile nodes cannot all be assumed to remain within wireless range of every other node. Our real-time protocol must therefore be capable of being instantiated at multiple points and each of these instances must be able to work with the others for the end-to-end setup/control of multihop real-time periodic streams.

  Multiple segment wireless networks create a number of new problems and one is the coordination of traffic in neighboring segments. If uncoordinated, simultaneous transmissions interfere with each; after all, unlike wired networks, wireless networks (whether mobile or not) do not have clear-cut traffic partitioning bridge nodes. Another problem comes up in admission control. We mentioned in chapter 4 that when a new periodic stream request is made, the current scheduler picks a suitable next scheduler host and sends the task set to it. Until the determination of the schedulability of this next task set, it is possible to receive additional admission requests. This is an event that would not occur very often in the single segment case. However, when we consider the multi-segment version of the problem and focus on a segment at a central region through which many multi-hop connections will have to go, then the likelihood of receiving multiple new stream requests almost simultaneously increase. Given the presence of multiple hosts in that segment, we may be able to consider subsets of the outstanding new requests for schedulability (for example, for three new requests, there would be $2^3 = 8$ potential schedules). If the number of hosts in this segment is less than the number of subsets, then some of the subsets may have to be dismissed.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Otherwise, each host can be used to explore the schedulability of one particular new set of tasks and the scheduler can then pick a new schedule that accommodates the most number of new streams.

- The completion of the multi-segment version of the protocol will allow us to use actual application processes to gather performance figures (the results we reported in chapter 3, 5, and 6 were based on randomly generated task sets). This is the third major benefit of using JiST: it allows basic networking applications written in Java to run within the simulation system in unmodified form.

- Finally, we plan to integrate the bandwidth allocation component of the real-time networking facilities to augment existing MANET routing protocols such as DSR [JM96] and AODV [Per97]. In this aspect, we have two specific goals. First, bandwidth allocations will allow these protocols to work in a more timely manner. Second, using upper bounds on the timeliness of routing updates, we may be able to quantitatively define upper bounds on the size of a MANET beyond which we can assume the network to have grown "too much for its own good".



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# Bibliography


[AB98] Alia K. Atlas and Azer Bestavros. Statistical rate monotonic scheduling. In *Proceedings of the Real-Time Systems Symposium - 1998*, 1998.

[Bar95] S. Baruah. Fairness in periodic real-time scheduling. In *Proceedings of the Real-Time Systems Symposium - 1995*, pages 200–209, 1995.

[BG91] Dimitri P. Bertsekas and Robert Gallager. *Data Networks.* Prentice Hall, second edition, 1991. ISBN: 0132009161.

[BL98] Sanjoy K. Baruah and Shun-Shii Lin. Pfair scheduling of generalized pinwheel task systems. *IEEE Transactions on Computers*, 47:812–816, 1998.

[BSS95] Giorgio Buttazzo, Marco Spuri, and Fabrizio Sensini. Value vs. deadline scheduling in overload conditions. In *Proceedings of the Real-Time Systems Symposium - 1995*, pages 90–99, 1995.

[Bur91] A. Burns. Scheduling real-time systems: A review. *Software Engineering Journal*, 6(3):116–128, 1991.

[BW91] A. Burns and A. J. Wellings. Priority inheritence and message passing: A formal treatment. *Real Time Systems*, 1991.

[CC89] H. Chetto and M. Chetto. Some results on the earliest deadline scheduling algorithms. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, 1989.

[CC92] M. Y. Chan and F. Chin. General schedulers for the pinwheel problem based on double integer reduction. *IEEE Transactions on Computers*, 41(6):755–768, jun 1992.

[CC93] M. Y. Chan and Francis Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica*, 9(5):425–462, 1993.

[cCV97a] Tzi cker Chiueh and Chitra Venkatramani. Design and implementation of a real-time switch for segmented ethernets. In *International Conference on Network Protocols (ICNP)*, 1997.

[cCV97b] Tzi cker Chiueh and Chitra Venkatramani. Fault tolerance mechanisms in the rether protocol. Technical report, University of New York, Stony Brook, 1997.




Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[Cin75] Erhan Cinlar. *Introduction to Stochasitc Processes.* Prentice Hall, 1975. ISBN: 0-13-498089-1. This book is quite a seminal introductory text to the field of stochastic processes. It is not as easy to understand as [RMF96], but the coverage is more extensive. I consider it a good follow up book after a first exposure to the sotchastic processes by [RMF96].

[DW95] Robert Davis and Andy Wellings. Dual priority scheduling. In *Proceedings of the Real-Time Systems Symposium - 1995*, pages 100–109, 1995.

[ER03] Ali Erkan and Zac Rider. Deterministic media access and bandwidth allocations for periodic streams. OPNETWORK, 2003.

[FHH+02] Hanssen F, P. Hartel, T. Hattink, P. Jansen, J. Scholten, and J.Wijnberg. A real-time ethernet network at home. *???*, 2002.

[Gal91] D. Le Gall. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):47–58, April 1991.

[GS88] J. Goodenough and L. Sha. Priority ceiling protocol: A method for minimizing the blocking of high-priority ada tasks, the. Technical Report CMU/SEI-88-SR-4, Software Engineering Institute (Carnegie Mellon University), 1988.

[HK95] Herman Hughes and Marwan Krunz. A traffic model for MPEG-Coded VBR streams. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, pages 47–55, New York, NY, USA, May 1995. ACM Press.

[HL89] Ching-Chih Han and Kwei-Jay Lin. Scheduling jobs with temporal distance constraints. 1989.

[HL92] C. C. Han and K. J. Lin. Scheduling distance constrained real-time tasks. In *Proceedings of the Real-Time Systems Symposium - 1992*, pages 300–308, 1992.

[HLF95] C. W. Hsueh, K. J. Lin, and N. Fan. Distributed pinwheel scheduling with end-to-end timing constraints. In *Proceedings of the Real-Time Systems Symposium - 1995*, pages 172–181, 1995.

[HLH96a] C.C. Han, K.J. Lin, and C.J. Hou. Distance-constrained scheduling and its applications in real-time systems. *IEEE Transactions on Computers*, 1996.

[HLH96b] C.C. Han, K.J. Lin, and C.J. Hou. Distance-constrained scheduling and its applications in real-time systems. *IEEE Transactions on Computers*, 1996.

[HLL95] Ching-Chih Han, Kwei-Jay Lin, and Jane W.-S. Liu. Scheduling jobs with temporal distance constraints. *SIAM Journal on Computing*, 24(5):1104–1121, October 1995.

[HMR+89] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: A real-time scheduling problem. In *Hawaii International Conference on System Sciences, Hicss-22, 1989, Vol 1 : Architecture Track*, pages 693–702, 1989.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[HRTV92] R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel. Pinwheel scheduling with two distinct numbers. *Theoretical Computer Science*, 100(1):105–135, 1992.

[HS95a] C. Han and K. Shin. Real-time communication in fieldbus multiaccess networks. In *Proc. of IEEE RTAS'95*, pages 86–95, May 1995.

[HS95b] C.-C. Han and K. G. Shin. A polynomial-time optimal synchronous bandwidth allocation scheme for the timed-token MAC protocol. In *Proceedings of the 14th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'95)*, pages 875–882, Los Alamitos, CA, USA, June 1995. IEEE Computer Society Press.

[JM96] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[KS95] Gilad Koren and Dennis Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *Proceedings of the Real-Time Systems Symposium - 1995*, pages 110–117, 1995.

[KSH95] M. Krunz, R. Sass, and H. Hughes. Statistical characteristics and multiplexing of MPEG streams. In *Proceedings of the 14th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'95)*, pages 455–462, Los Alamitos, CA, USA, June 1995. IEEE Computer Society Press.

[KSS95] Daniel I. Katcher, Shirish S. Sathaye, and Jay K. Strosnider. Fixed priority scheduling with limited priority levels. *IEEE Transactions on Computers*, 44(9):1140–1144, sep 1995.

[Leh90] J. P. Lehoczky. Fixed priority scheduling of periodic tasks with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–209, December 1990.

[LGW03] Alberto Leon-Garcia and Indra Widjaja. *Communication Networks*. McGraw-Hill Science/Engineering/Math, 2 edition, 2003.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinary*, 20(1):46–61, jan 1973.

[LL97] C. Lin and K. J. Lin. A pinwheel scheduler for three distinct numbers with a tight schedulability bound. *Algorithmica*, 19(4):411–426, 1997.

[LS86] J. P. Lehoczky and L. Sha. Performance of real-time bus scheduling algorithms. *ACM Performance Evaluation Review*, 14, 1986.

[LS04] Amy N. Langville and William J. Stewart. The kronecker product and stochastic automata networks. *Journal of Computational and Applied Mathematics*, 167:429 – 447, 2004.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[LSD89] I. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium - 1989*, pages 166–171, 1989.

[LW82] J. Leung and J. Whitehead. On the complexity of fixed priority scheduling of periodic, real-time tasks. *Performance Evaluaiton*, 2(1):237–250, 1982.

[MBCG98] Aloysius Mok, Sanjoy Baruah, Deji Chen, and Sergey Gorinsky. Real-time scheduling of multimedia tasks. Technical Report CS-TR-98-14, University of Texas, Austin, May 1, 1998.

[MC96a] Aloysius K. Mok and Deji Chen. A general model for real-time tasks. Technical Report CS-TR-96-24, University of Texas, Austin, October 3, 1996.

[MC96b] Aloysius K. Mok and Deji Chen. A multiframe model for real-time tasks. Technical Report CS-TR-96-07, University of Texas, Austin, April 1, 1996.

[Mok83] A. K. Mok. *Fundamental Design Problems of Distributed Systems for hard real-time environments.* PhD thesis, MIT, 1983.

[PC98] Prashant Pradhan and Tzi-Cker Chiueh. Real time performance guarantees wired wired wireless lans. Technical report, University of New York, Stony Brook, 1998.

[Per97] C. Perkins. Ad-hoc on-demand distance vector routing, 1997.

[PeZ94] Pramod Pancha and Magda el Zarki. Mpeg coding for variable bit rate video transmission. 1994.

[RMF96] Ciriaco Valdez (Contributor) Richard M. Feldman. *Applied Probability and Stochastic Processes.* PWS Pub Co;, 1996.

[RR97] Theodore H. Romer and Louis E. Rosier. An algorithm reminiscent of euclidean-gcd computing a function related to pinwheel scheduling. *Algorithmica*, 17(1):1–10, 1997.

[RS84] K. Ramamritham and J. A. Stankovic. Dynamic task scheduling in hard real-time systems. 1(3):65–77, 1984.

[SG90] Lui Sha and John B. Goodenough. Real-time scheduling theory and Ada. *Computer*, 23(4):53–62, April 1990.

[SG94] Madalene Spezialetti and Rajiv Gupta. Exploiting program semantics for efficient instrumentation of distributed event recognitions. pages 181–190. Symposium on Reliable Distributed Systems, 1994.

[SLR87] L. Sha, J. P. Lehockzy, and R. Rajkumar. Task scheduling in distributed real-time systems. In *Proceedings of IEEE Industrial Electronics Conference*, 1987.

[SR85] J. A. Stankovic and K. Ramamritham. Evaluation of a flexible task scheduling algorithm for distributed hard-real time systems. C-34(12):1130–1143, 1985.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[SRL90] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronisation. *IEEE Transactions on Computers*, 39(9):1175–1185, sep 1990.

[SRLR89] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):243–265, 1989.

[SS94] Lui Sha and Shirish S. Sathaye. *Advances in Real-Time Systems*, chapter A Systematic Design Approach to Designing Distributed real-Time Systems, pages 149–170. Prentice Hall, 1994.

[SSDB95] John A. Stankovic, Marco Spuri, Marco Di Natale, and Giorgio C. Buttazzo. Implications of classical scheduling results for real-time systems. *IEEE Computer*, 28(6):16–25, June 1995.

[TBW92] K. W. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority preemptively scheduled systems. In *Proceedings of the Real-Time Systems Symposium - 1992*, pages 100–109, 1992.

[VcC95] Chitra Venkatramani and Tzi cker Chiueh. Acm sigcomm. In *Design, Implementation, and Evaluation of A Software-Driven Real-Time Ethernet Protocol*, 1995.

[War91] Carl Warren. Rate monotonic scheduling. *IEEE Micro*, 1991.

[wHL01] Chih wen Hsueh and Kwei-Jay Lin. Scheduling real-time systems with end-to-end timing constraints using the distributed pinwheel model. *IEEE Transactions on Computers*, 50(1):51–66, 2001.

[WL83] W. D. Wei and C. L. Liu. On a periodic maintenance problem. *Operations Research*, 2(2):90–93, 1983.

[WSG98] Wanqing Wu, Madalene Spezialetti, and Rajiv Gupta. A protocol for removing communication intrusion in monitored distributed systems. pages 120–129. ICDCS, 1998.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

# Vita

Ali Erkan was born in Istanbul Turkey in 1967 to Metin and Selma Erkan. He received a B.S. degree in Computer Engineering (1991) and an M.S. degree in Computer Science (1995) from Lehigh University. During his graduate studies, he worked on projects in distributed monitoring, parallel numerical computing, and real-time scheduling. After a three year temporary appointment at Swarthmore College (2001-2004), he started a tenure-track position at Ithaca College as an assistant professor of Computer Science (2004-).

**Courses Taught At Ithaca College**: Principles of Computer Science I, Discrete Math, Data Structures, Computer Networks, Algorithms.

**Courses Taught At Swarthmore College**: C & Unix: The Imperative Paradigm, Algorithms and Object-Oriented Programming, Advanced Algorithms, Computer Network Modeling, Computer Network Programming, Computer Networks.

**Publications With Undergraduates**:

Ali Erkan and Zac Rider. *Deterministic Media Access and Bandwidth Allocation for Periodic Streams.* OPNETWORK, August 2003, Washington DC.

Ali Erkan and Joshua Hudner. *A Simulation Based Study of the Allen-Cunneen Approximation in Queuing Networks with non-Markovian Traffic Sources.* OPNETWORK, August 2003, Washington DC.

**Awards, Grants, Accomplishments**

Summer Research Grant at Swarthmore College (Summer 2003), FIGLEAF grant, Swarthmore College (Spring 2002), Fellowship, EECS Department, Lehigh University (Spring 1996), Arthur E. Humphrey Teaching Assistant Award, Lehigh University (Spring 1994).



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.