Technical Report

CMU/SEI-93-TR-002
ESC-TR-93-179

Distributed Real-Time System Design: Theoretical Concepts and Applications

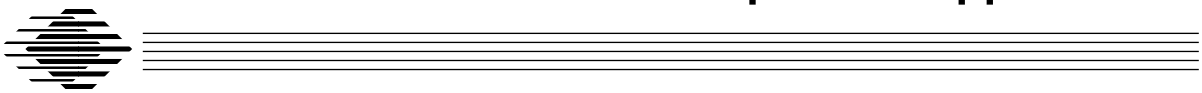
Lui Sha
Shirish S. Sathaye

March 1993

Technical Report

CMU/SEI-93-TR-002
ESC-TR-93-179
March 1993

Distributed Real-Time System Design: Theoretical Concepts and Applications



Lui Sha

Dependable Real-Time Software Project

Shirish S. Sathaye

Department of Electrical and Computer Engineering
Carnegie Mellon University

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212.
Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145. Phone: (703) 274-7633.

Table of Contents

List of Figures

List of Tables

Distributed Real-Time System Design: Theoretical Concepts and Applications

Abstract: Distributed real-time system design raises new theoretical issues and application challenges, beyond those of centralized systems. Rate monotonic scheduling (RMS) theory has been successfully applied in the scheduling of centralized systems. RMS and its generalizations have been adopted by national high technology projects such as the Space Station and has recently been supported by major open standards such as the IEEE Futurebus+ and POSIX.4. In this paper, we describe the use of generalized rate monotonic scheduling theory for the design and analysis of a distributed real-time system. We review the recent extensions of the theory to distributed system scheduling, examine the architectural requirements for use of the theory, and finally provide an application example.

1. Introduction

Real-time computing systems are critical to an industrialized nation's technological infrastructure. Modern telecommunication systems, factories, defense systems, aircraft and airports, space stations and high energy physics experiments cannot operate without them. Indeed, real-time computing systems control the very systems that keep us productive, safeguard our liberty, and enable us to explore new frontiers of science and engineering.

In real-time applications, the correctness of a computation depends upon not only its results but also the time at which outputs are generated. The measures of merit in a real-time system include:

- Predictably fast response to urgent events.
- High degree of schedulability. Schedulability is the degree of resource utilization at or below which the timing requirements of tasks can be ensured. It can be thought as a measure of the number of[timely transactions per second.
- Stability under transient overload. When the system is overloaded by events and it is impossible to meet all the deadlines, we must still guarantee the deadlines of selected critical tasks.

Real-Time scheduling is a vibrant field. Generalized Rate Monotonic theory (GMRS), one of several important research efforts ([10], [11]), is a useful tool that allows system developers to meet the above requirements by managing system concurrency and timing constraints at the level of tasking and message passing. In essence, this theory ensures that all tasks meet their deadlines as long as the system utilization of all tasks lies below a certain bound, and appropriate scheduling algorithms are used. This puts the development and maintenance of real-time systems on an analytic, engineering basis, making these systems easier to develop and maintain.

Application of this theory to centralized system scheduling is well known [5]. The focus of this paper is on recent generalization of this theory to schedule large scale distributed real-time systems. In addition, we provide an example that illustrates some of the practical development problems in the application of this approach in actual system development: to address the increasing trend towards open system components and to control costs, system designers have to use standardized subsystem components, which may not support real-time computing needs. To reduce the number of processors in a system, sometimes both real-time and non-real-time applications must co-exist in a processor and share the network with real-time traffic.

The following is a high level view of our example application, which will be presented and solved in detail later in this paper. Since a widely available standard network that supports GRMS does not currently exist, we build our example system around the ANSI X3T9.5 FDDI network as shown in Figure 1-1. Since IEEE Futurebus+ (896) and POSIX.4a support the use of GRMS for real-time applications, we use them in our example as the station backplane and operating system respectively. From the application view point, our example consists of both classical real-time surveillance and control applications, and multimedia applications.

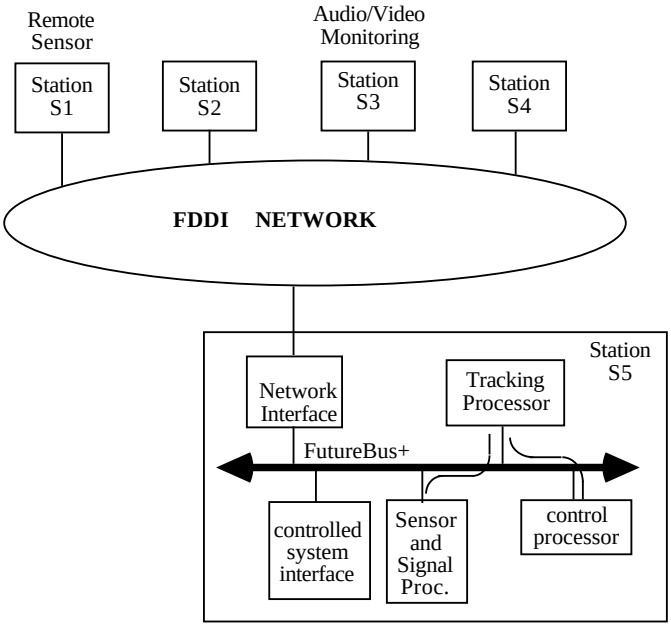


Figure 1-1: Block Diagram of Distributed Real-Time System

The rest of the paper is organized as follows. Chapter 2 presents a synopsis of generalized rate monotonic theory for centralized systems. Chapter 3 describes the theoretical extensions necessary for applying GRMS to a distributed system. Chapter 4 introduces the notion of scheduling abstractions, a technique for using and analyzing existing subsystems

that are not designed to support GRMS. This is a very important aspect in the application of this theory to real-world systems. Chapter 5 describes a comprehensive example that illustrates task scheduling within subsystems as well as end-to-end scheduling in a large real-time system. Finally we make concluding remarks in Chapter 6.

2. Synopsis: GRMS in Centralized Systems

A real-time system typically consists of both periodic and aperiodic tasks. A periodic task τ_i is characterized by a worst case computation time C_i and a period T_i. Unless mentioned otherwise we assume that a periodic task must finish by the end of its period. Tasks are *independent* if they do not need to synchronize with each other. By using either a simple polling procedure or a more advanced technique such as a sporadic server [9], the scheduling of aperiodic tasks can be treated within the rate monotonic framework. In each case C units of computation are allocated in a period of T for aperiodic activity. However, the management and replenishment of the capacity is different in each case.

The scheduling of independent periodic tasks was originally considered by Liu and Layland [4]. The scheduling of periodic tasks with synchronization requirements can be addressed by a simple extension to the original formula as follows [6]:

> **Theorem 1:** A set of *n* periodic tasks scheduled by the rate monotonic algorithm will always meet its deadlines, for all task phasings, if
>
> $$\forall\, i,\ 1 \le i \le n, \quad \frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{(C_i + B_i)}{T_i} \le i(2^{1/i} - 1)$$

where B_i is the duration in which task τ_i is blocked by lower-priority tasks. This blocking which is also known as priority inversion. The effect of this blocking can be modeled as though task τ_i's utilization is increased by an amount B_i/T_i. Theorem 1 shows that the duration of priority inversion reduces *schedulability*, the degree of processor utilization at or below which all deadlines can be met. Priority inversion in a centralized system can occur when tasks have to synchronize and have common critical sections. This inversion can be controlled by a *priority ceiling protocol*. The *priority ceiling protocol* is a real-time synchronization protocol described in detail in Sha, Rajkumar, and Lehoczky [6]. Under this protocol there are no mutual deadlocks, and a higher-priority task can be blocked by lower-priority tasks only once. As a result, the worst case duration of blocking is the longest critical section that may block a task.

A tutorial on GRMS that contains further details on these issues is given in Sha and Goodenough [5].

3. Distributed System Extensions for GRMS

Scheduling in a network is different from scheduling in a centralized environment. In a centralized system, the centralized scheduler immediately knows of all resource requests. In some networks, distributed scheduling decisions must be made with incomplete information. From the perspective of any particular station, some requests could be delayed and some may never be seen, depending on the relative position of the station in the network. The challenge is to achieve predictability under these circumstances.

GRMS theory has to be extended to address this challenge. Certain basic concepts such as schedulability and preemption need to be revisited, and some concepts such as system consistency need to be developed.

3.1. Extensions to Schedulability Concept

In a real-time system a particular activity is said to have "met its deadline" if the activity completes by its deadline. When scheduling tasks on a processor, each task is said to have met its deadline if it completes execution by a certain time before the end of its period. In a communication network, the delay incurred by a message in reaching its destination is the sum of the transmission delay and the propagation delay. The transmission delay is the time between message arrival at a station and the time at which it is transmitted. The transmission delay can be treated analogously to task execution on a processor. However, the propagation delay can be longer than packet transmission times causing the transmission of the next message to begin before a particular message reaches its destination. This occurs in networks such as FDDI, IEEE 802.6 distributed-queue dual-bus (DQDB), and even IEEE 802.5 token rings when early token release is used. It is therefore useful to separate transmission delay and propagation delay and consider the notion of *transmission schedulability* [7]. A set of messages is said to be *transmission schedulable* (t-schedulable) if each message can be transmitted before its deadline. Satisfaction of the end-to-end deadline of the message can be found using the relation:

$$End\text{–}to\text{–}End\ Deadline \geq Transmission\ Deadline + PropagationDelay$$

For example, in an FDDI network, the worst case propagation delay is the *walk time*, defined as the time taken by a single bit to traverse the ring if no station on the ring wanted to transmit.

3.2. Preemption Control

From the user's viewpoint, a certain initial delay in setting up a periodic connection is acceptable. However, users expect a steady flow of information once the connection is set up, and hence require that C packets be delivered every period T. We will discuss the need for preemption control to achieve the above property.

Preemption is the most basic concept in priority scheduling. Tasks are assigned priorities according to some algorithm to maximize resource (e.g., processor) utilization. It has been a long held belief that the idealized form of priority scheduling is to achieve instantaneous preemption, i.e., whenever a high-priority task becomes ready, the resource is immediately taken away from any lower-priority task and given to the high-priority task.

It has been assumed that increasing preemptability always leads to a minimization of priority inversion, and that priority inversion is eliminated if a higher-priority task can always preempt a lower priority task. However, this is not true in a distributed system. In a distributed system there can be special situations when a particular preemption increases the delay experienced by lower-priority connections, but does not reduce the worst case duration of priority inversion. We call such situations *over-preemption*, and their effect is to reduce the schedulable utilization of the network. To overcome the undesirable effect of over-preemption, a *preemption control* protocol is needed. In the following, we use a dual-link network based on the IEEE 802.6 DQDB [8] as an example to introduce the two aspects of our preemption control protocol, namely *phase control* and *rate control*. Finally, we will address the logical relation between preemption control and the priority inversion.

IEEE 802.6 DQDB Operation

The IEEE 802.6 DQDB MAC [1] protocol specifies a pair of slotted links operating in opposite directions. The links may be referred to as Flink and Rlink respectively as shown in Figure 3-1.

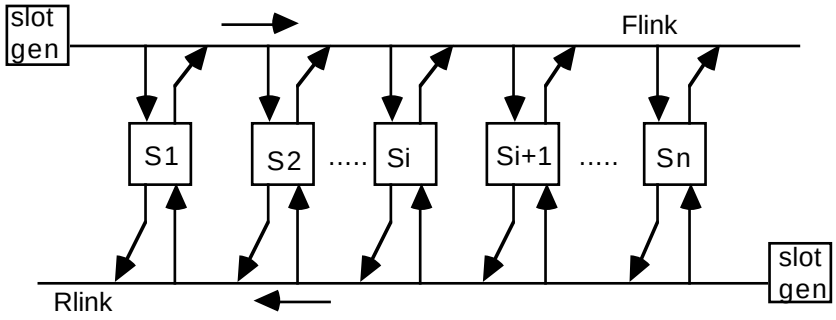


Figure 3-1: IEEE 802.6 DQDB Network

Fixed-length slots are generated by slot generators of the corresponding links. Although the figure shows slot generators as separate functional units, the slot generation function can be embedded in stations at the end of the links. Each station is able to transmit and receive messages on both links. The selection of the link to be used for transmission depends on the physical location of the destination. Reservation for a slot on the Flink is made on the Rlink via a **request** and vice versa.

The operation of the protocol is based on a single busy bit, indicating whether the slot is used or free, and a request bit per slot for each priority level. Four priority levels are supported. Each priority level represents a separate access queue. A station wishing to transmit at a certain priority on Flink, issues a

request in a slot on Rlink by setting the proper request bit. It also places its own request into its access queue at the correct priority. Each station on seeing a request, enqueues it in its access queue at the correct priority. Every station on seeing a free slot discards the top request from its highest priority non-empty access queue, because the slot has been previously reserved by another station. If the top request is the station's request then it transmits in the slot on the Flink in addition to removing the request from its access queue. The access queues are implemented using a set of two counters for each priority level. Details may be found in [1].

The current IEEE 802.6 protocol does not have adequate mechanisms to ensure correct operation in a real-time distributed scheduling environment [3]. As a result, it exhibits unpredictable behavior under certain conditions [2,3]. The distributed scheduling extension to GRMS reported in this paper has been the result of studying these problems. Finally, IEEE 802.6 implements only 4 priority levels. As we will see in Chapter 4, this also results in a reduction of schedulability. However, IEEE 802.6 provides high bandwidth over a long distance and the problems mentioned here can be solved by extensions to the standard [3] or by the development appropriate scheduling abstractions.

References

1. IEEE 802.6 Distributed Queue Dual Bus - Metropolitan Area Network, Draft Standard, Version P802.6/D15, October 1990.
2. Van As, H. R.; Wong, J. W.; and Zafiropulo, P. "Fairness, Priority and Predictability of the DQDB MAC Protocol Under Heavy Load", *Proceedings of the International Zurich Seminar*, March 1990, pp. 410-417.
3. Sha, L.; Sathaye, S.; and Strosnider, J. "Scheduling Real-Time Communication on Dual Link Networks", *13th IEEE Real-Time Systems Symposium*, December 1992.
4. Van As, H. R.; Wong, J. W.; and Zafiropulo, P. "Fairness, Priority and Predictability of the DQDB MAC Protocol Under Heavy Load", *Proceedings of the International Zurich Seminar*, March 1990, pp. 410-417.
5. Sha, L.; Sathaye, S.; and Strosnider, J. "Scheduling Real-Time Communication on Dual Link Networks", *13th IEEE Real-Time Systems Symposium*, December 1992.

Example 1: Consider a dual-link network with three stations, as shown in Figure 3-2. Let the delay between S_1 and S_2 be 10 slots and between S_2 and S_3 be 1 slot as shown. Let S_1 and S_3 be transmitting as follows: S_1 has a low-priority connection that uses 100 slots every 10000 slots. S_3 has a medium priority connection that wants to transmit in 1 slot every 10 slots. This leads to a slot usage pattern as shown. Slots labeled **L** are used by S_1, and the slot labeled **M** is used by S_3. Notice that S_1 has released an empty slot so that S_3 may transmit once every 10 slots are it requires. Now let S_2 start a new high-priority connection that needs to transmit in 1 slot every 4 slots. Since S_2's request has higher priority, it preempts S_3's request in S_2's queue and S_2 will transmit in the unused slots that were meant for S_3. The first of the slots released by S_1 for S_2 will take 20 units of time after S_2's first request to reach S_2. Until this time since S_2 can only transmit in slots meant for S_3, S_2 can transmit only one slot in 10 which is less than it needs. As a result, even though S_3's connection is interrupted, S_2 is not t-schedulable, resulting in an erratic connection. Therefore the preemption of S_3's request is a form of over-preemption.

To correct the problem in the above example we need to prevent Station S_2 from using slots

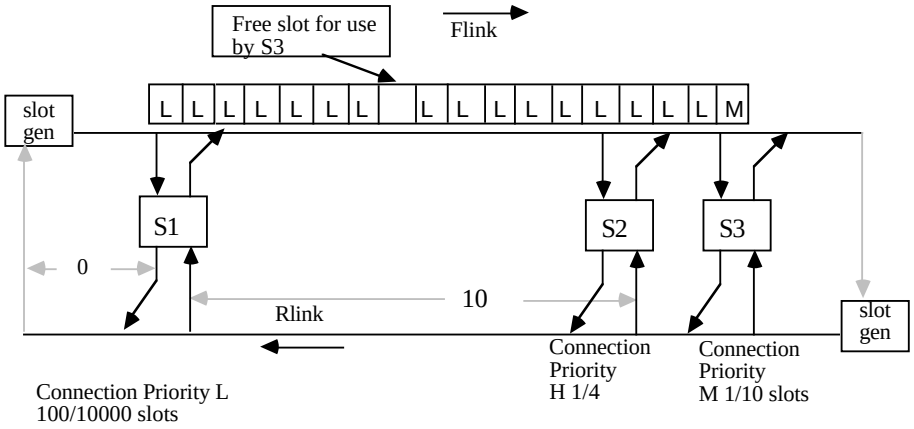


Figure 3-2: Preemption Control Example

released for station S_3. This means that S_2 should delay its slot use for 20 slot times after its first request, which is the round trip delay between S_2 and the slot generator for the Flink. After this time, slots released by S_1 in response to S_2's request will reach S_2. This is the phase control aspect of preemption control.

However, phase control by itself is insufficient. During the 20 unit delay 5 cells are buffered at S_2. After the 20 unit delay, only 1 slot in 4 will be released for use by S_2. Hence S_2 attempts to transmit all 5 slots as soon as possible, then connection from S_3 will again be disrupted without improving S_2's worst case end-to-end latency. Observe that the 20 unit delay will add to S_2's worst-case delay irrecoverably. After the connection is set-up, the destination expects 1 cell every 4 slots; therefore, attempting transmission of all 5 cells as soon as possible does not improve S_2's worst-case performance. Hence, S_2 should only transmit one slot every 4, after the round-trip delay of 20 slot times. This is the rate control aspect of preemption control. With phase and rate control, the connections for both S_2 and S_3's will be *transmission schedulable*.

Finally, we want to point out that from an implementation viewpoint, the preemption control occurs at a higher layer than the priority queueing mechanism. Prioritized packets released by preemption control protocol into the MAC layer will follow usual priority queueing rules as in a centralized system.

3.3. System Consistency

As discussed before, stations in a distributed system may have incomplete or delayed information of the system state. This may lead to inconsistent views of the system state, as illustrated by the following example:

Example 2: Consider three stations S_c, S_b, and S_a, in a dual-link network as shown in

Figure 3-3. Suppose S_b enters its own request R_b in its transmission queue and then attempts to make a request on the Rlink. Let S_b be prevented from making a request on Rlink by higher-priority requests until request R_a by station S_a passes by. On the request stream, R_a precedes R_b while in S_b's transmission queue R_b precedes R_a. After the requests are registered in station S_c, the transmission queue of S_c will have R_a preceding R_b, which is inconsistent with the queue of station S_b.

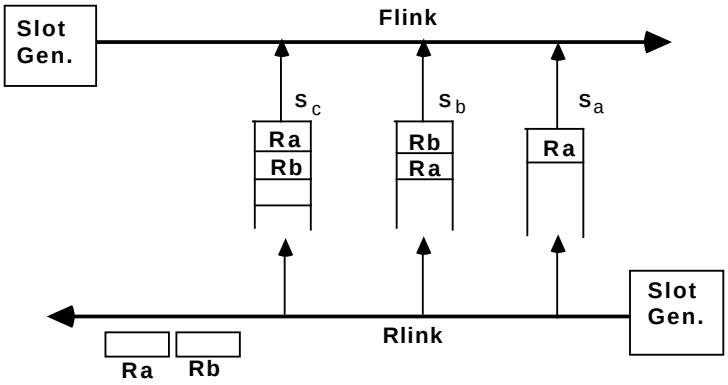


Figure 3-3: Inconsistent Station Queues in an IEEE 802.6 DQDB Network

To address this problem we introduce the concept of system consistency. System consistency can be defined as follows: In a distributed system it is possible for two request entries to exist in multiple queues. For example, two requests can simultaneously exist in multiple station queues in a dual-link network [1]. A system is said to be *consistent* if and only if the order of the same entries in different station queues is consistent with each other. For example in a dual-link network, if request R_1 and request R_2 both exist in queue Q_a and queue Q_b, and if R_1 is ahead of R_2 in Q_a, then R_1 must also be ahead of R_2 in Q_b.

The inconsistency problem can lead to conflicts between distributed scheduling actions. Inconsistency can be avoided by the following rule: A station is not permitted to enter its request in its own queue until it has successfully made the request on the link. This makes the entries in each queue consistent with the ordering of requests on the link. Therefore all the queues will be consistent with each other. In the above example, station S_b cannot enter its request in its queue until it can make a request on the link. Therefore, S_b's request will be after S_a's request, both on the link and in S_b's queue.

In the preceding paragraphs we have highlighted fundamental new issues in distributed real-time system design. These issues are intrinsic to a wide area network where communication delays are long and scheduling has to be carried out in parallel by distributed stations with partial or delayed information. Any distributed real-time system protocol must address at least some of these issues.

A formal description of the above concepts is described as a coherent reservation protocol (CRP) and a preemption-control protocol [8]. The important theoretical result of this work is summarized by the following theorem:

> **Theorem 1:** For a given a set of periodic connections in a dual-link network that follows CRP, if the set of connections is schedulable in a centralized preemptive priority-driven system with zero (negligible) propagation delay, then the set of connections is transmission schedulable in a dual-link network.

The importance of this theorem is that even in a wide area network with incomplete information, scheduling decisions can be made as though it is a centralized system. This allows us to seamlessly use GRMS in the analysis of such systems.

4. Scheduling Abstractions

GRMS assumes preemptive, priority scheduling. Ideally, each distinct period corresponds to a distinct priority level. However, only a limited number of priorities can be supported by hardware. The effect of limited priority levels is reduced schedulability as illustrated by the following diagram [7]:

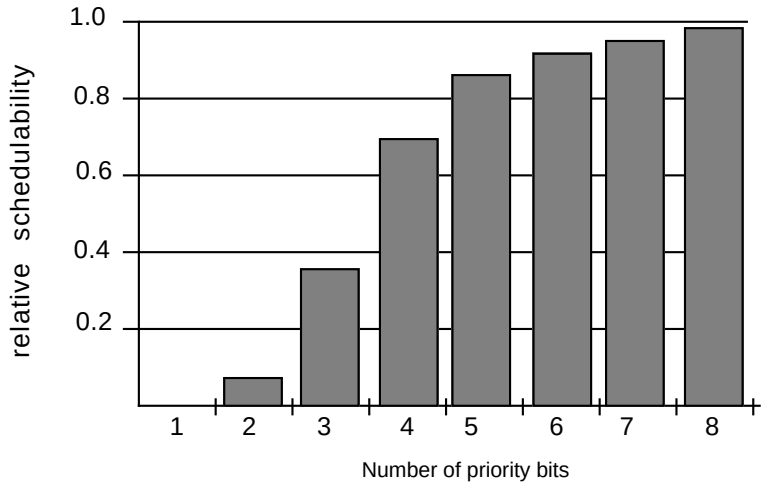


Figure 4-1: Schedulability Loss vs. The Number of Priority Bits

Figure 4-1 plots the schedulability as a function of priority bits, relative to the schedulability with as many priority levels as needed under the condition that the ratio between the largest and the shortest period is 100,000 [7]. As can be seen, the schedulability loss is negligible with 8 encoded priority bits, which corresponds to 256 priority levels. In other words, the worst-case schedulability obtained with 8 priority bits is close to that obtained with an unlimited number of priority levels.

To use GRMS for the development of real-time computing systems, we would like to use subsystems that support the use of GRMS such as Futurebus+, POSIX.4 and Ada 9x. However, we may have to use some components that do not support GRMS. In this case, we need to develop a *scheduling abstraction* for the sub-system so that it can be treated as if it supports GRMS. Although the scheduling abstraction allows the use of GRMS, it comes at cost of reduced schedulability due to the lack of direct support. With scheduling abstractions, we can provide application developers a consistent scheduling interface that allows them to develop applications as if every sub-system supports GRMS. In the following, we demonstrate the creation of scheduling abstractions by using the FDDI timed token protocol as an example.

Fiber Distributed Data Interface

FDDI is a 100 Mbit/s Local/Metropolitan Area Network that has gained recent popularity. FDDI is a token ring protocol that uses a timed-token access method [1]. In a token rotation media access protocol, stations are connected to form a ring. All messages move around the ring and are repeated by each station through which they pass. A station reading its own address as the destination copies the packet and then passes the packet to the next station in the ring. Once the frame reaches the source station, it is removed from the ring. The permission to transmit is granted to a station that is in possession of a special type of frame called a *token*. The time for a token to traverse an idle ring is called the *walk time*, denoted here as W_T.

Under this protocol, stations on the network choose a *target token rotation time* (TTRT). A station in the FDDI protocol can transmit in either synchronous or asynchronous mode. Each station is allocated a *synchronous capacity*, which is the maximum time a station is permitted to transmit in synchronous mode every time it receives the token. Synchronous capacities of each station are restricted to a pre-allocated fraction of $(TTRT - W_T)$, such that the cumulative synchronous capacity of the entire network is bounded by $(TTRT - W_T)$. When a station receives a token, it first transmits its synchronous traffic for an amount of time bounded by its synchronous capacity. Then it may transmit asynchronous traffic only if the the time since the previous token departure from the same station is less than TTRT. This protocol forces the token to rotate at a speed such that the time between two consecutive token visits is bounded by 2*TTRT [2]. In a network that uses only synchronous mode, time between consecutive token arrivals is bounded by one TTRT.

References

1. FDDI Token Ring Media Access Control -- ANSI Standard X3T9.5/83-16, 1987.
2. Sevcik, K. C. and Johnson, M. J. "Cycle Time Properties of the FDDI Token Ring Protocol", *IEEE Transactions on Software Engineering*, SE-13, No. 3, 1987, pp 376-385.

A real-time scheduling analysis of FDDI for the case of one periodic connection per station has been developed [2]. In this paper, using the normalized proportional allocation scheme in Agrawal, Chen, Zhao, and Davari [2], we create a scheduling abstraction when there are more than one periodic connections per station. In the development of this abstraction, we need to consider priority granularity, priority inversion, system consistency, pre-emption control, and transmission schedulability.

In an FDDI network that uses only synchronous mode, each station S_i can transmit once every TTRT for an amount equal to an assigned synchronous capacity H_i. Therefore the resource (network) is allocated to stations in a time-division multiplexed fashion, with no priority between stations. As an example, Figure 4-2 shows the transmission sequence from three stations S_1, S_2, and S_3, allocated bandwidths of H_1, H_2, and H_3 respectively. Hence it may appear from Figure 4-1 that the schedulable utilization is zero. However, the order of message transmissions from each station may be prioritized. If each station implements sufficient priorities to use its dedicated portion of the bandwidth, then there is no schedulability loss within stations. However, since there is no priority arbitration between stations, a station with the token can transmit lower-priority messages even when high-priority messages are waiting. In this sense, it is a bounded priority inversion and limits the schedulable utilization of the network.

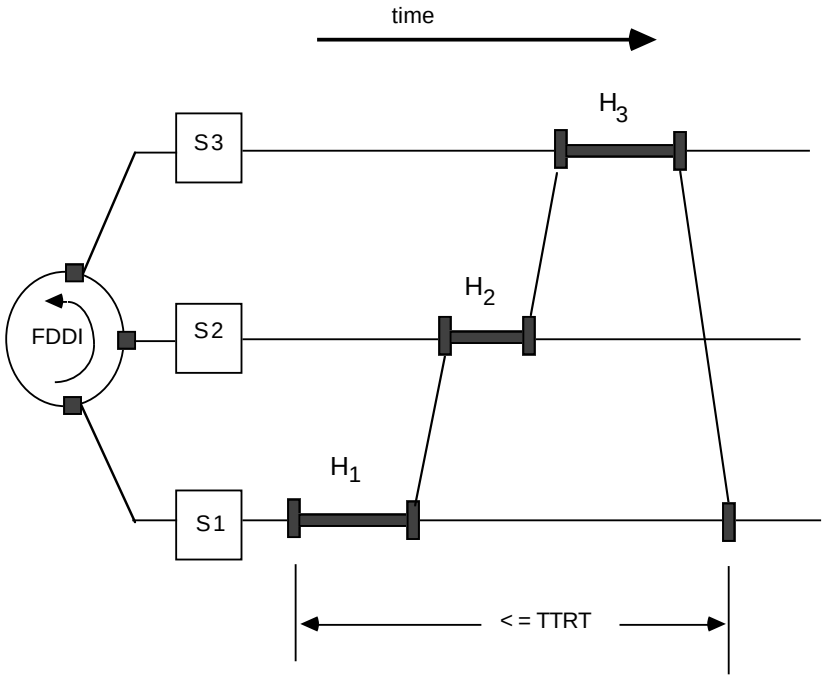


Figure 4-2: Transmission sequence from three stations in synchronous mode

A message must satisfy these conditions to be schedulable in an FDDI network that operates in synchronous mode:

- Each connection's period T_i must satisfy the relation $T_i \geq TTRT$.
- Each station S_i must be allocated enough synchronous capacity H_i so that each connection in the station is t-schedulable.

A simple scheme for synchronous bandwidth allocation is the normalized proportional scheme suggested by Agrawal, Chen, Zhao, and Davari [2]. The total available bandwidth on each token rotation is given by $(TTRT - W_T)$. The normalized proportional allocation scheme gives each station a fraction of this bandwidth, consistent with that station's contribution to the total network utilization. Therefore, the bandwidth H_i allocated to station S_i is given by:

$$H_i = \frac{U_i}{U}(TTRT - W_T)$$

U_i is the network bandwidth utilized by station S_i and $U = U_1 + \ldots + U_n$. $TTRT$ is the target

token rotation time and W_T is the walk time. The following example demonstrates this formula:

> Example 3: Suppose we have three periodic messages τ_1, τ_2 and τ_3, to be transmitted from three stations S_1, S_2, and S_3 respectively, on an FDDI network. Let the TTRT be 8 and the walk time W_T be 1.
>
> - Message τ_1: C_1 = 7; T_1 = 100
> - Message τ_2: C_2 = 10; T_2 = 145
> - Message τ_2: C_3 = 15; T_2 = 150
>
> where C_i is the transmission time and T_i is the period of message τ_i. The utilization of the above message set, U = 0.239. Applying the above formula, H_1 = 2.05, H_2 = 2.02, H_3 = 2.93.

Consider the application of GRMS to an FDDI network that transmits only in synchronous mode. Let synchronous bandwidths be allocated to stations by some technique such as the above formula. Notice that if each station uses its allocated synchronous bandwidth, the actual token rotation time (TRT) (time between consecutive token arrivals at the station) attains its maximum possible value TTRT. However if any station does not use its allotted bandwidth, then the token rotates faster than TTRT.

Consider any station S_i in the network. Let the capacity allocated to the station be H_i. Let the station be a source of periodic messages $\tau_{1i} = (C_{1i},T_{1i})$, $\tau_{2i} = (C_{2i},T_{2i})$, . . . , $\tau_{ni} = (C_{ni},T_{ni})$. The station can transmit for up to H_i units of time, every TTRT.

GRMS can be applied to scheduling messages in this station as follows: In addition to the message set that must be transmitted, the station can be considered to have another "message" to transmit. The period of this message is the actual token rotation time TRT. The "transmission time" of this message is given by $(TRT–H_i)$. We refer to this task as Token Rotation Message τ_{tr}. The token rotation message represents the time that the station is prevented from transmitting every token rotation. The longest period of the token rotation task is TTRT. A necessary condition for schedulability is that the period of the highest frequency message must be longer than TTRT. Note that the actual token rotation time can be shorter than TTRT if other stations do not completely use their synchronous allocations. However, station S_i is guaranteed H_i amount of bandwidth in every token rotation cycle. Hence, if connections in S_i are schedulable in the longest cycle (TTRT), they are also schedulable in any shorter cycle.

The FDDI scheduling abstraction described above has two levels of scheduling. At the higher level, the resource capacity is allocated between applications in a time division multiplexed (TDM) manner. Within each allocation, the resource schedules activities using GRMS. This abstraction can be directly used for sharing a processor between real-time and non-real-time applications. In this case, we create a cycle and allocate portions of the cycle to real-time and non-real-time activities respectively. Observe that similar to FDDI, the cycle has to be no greater in length than the period of the highest frequency real-time task. The TDM switching overhead can be treated similar to the walk-time in the FDDI case.

Finally, consistency between station queues is not an issue since each station has its own dedicated bandwidth. Preemption control is still necessary when a new connection has to be established and synchronous bandwidth has to be reallocated. In this case, the connection should first exercise phase control and avoid transmitting until bandwidth is allocated for the new connection. Furthermore, the new allocation should exercise rate control and not transmit all its accumulated packets. Finally, the concept of transmission schedulability is directly applicable in this abstraction as will be discussed in Chapter 5.

5. Example Application

We have reviewed the theory for applying GRMS to a distributed system and developed a scheduling abstraction for FDDI, the only component in our system that does not directly support GRMS. This example illustrates how to apply GRMS extensions to schedule a distributed system that consists of both real-time control activities and multimedia communication. This example will illustrate the following concepts:

- Management of end-to-end deadline by partitioning into subsystem deadlines
- Sharing of a processor by both real-time and non-real-time activities
- Application of the FDDI scheduling abstraction
- Management of propagation delay and jitter

5.1. Description of Example

Consider the system in Figure 1-1. It is built around an FDDI network. Station S_5 is a multiprocessor built around a Futurebus+ backplane. The control processor in station S_5 receives a variety of sensor, audio and video information, from both local and remote sources. There exists an end-to-end deadline from each source to the control processor. The end-to-end deadline is the permissible delay between the instant the information is captured at the sensor, to the instant the system outputs its response to the information. We assume that the priority ceiling protocol is used for task synchronization. A high-level description of the data flow in this example system is as follows. All time-units in the following example are milliseconds.

Traffic Across Network:

- *Station* S_1: Remote sensor information is captured and transmitted across the network to the control processor in Station S_5. The station transmits 1.0 Mbits of data every 150 . Also sends 5 Mbits of data every 100 to Stations 2 and 4.
- *Station* S_3: Video monitoring station captures audio and video information and transmits it over the network to the control processor in Station 5. The required end-to-end deadline is 100 . The video source is 1024x768 pixels per frame at 24 bits/pixel and 30 frames/sec. Three CD quality audio channels sampled at 44.1 Khz with 32 bits/sample are also transmitted.

Workload in Station S_5

- *Signal Processor Tasks*: The local sensor takes an observation every 40. To reduce unnecessary bus traffic the signal processing task processes the signal and averages it every 4 cycles before sending it to the tracking processor.
- *The tracking processor tasks:* After the task executes it sends the result to the control processor with a period of 160. Task τ_3 on the control processor uses this tracking information. In addition, the end-to-end latency of the pipeline of data flow from the sensor to the control processor should be no more than 785.
- *Control Processor Tasks:* The control processor has additional periodic and aperiodic tasks which must be scheduled.

- Aperiodic event handling with an execution time of 5 and an average inter-arrival time of 100.
- A periodic task for handling local feedback control with a computation requirement and a given period. Task τ_2: $C_2 = 78$; $T_2 = 150$;
- A periodic task that utilizes the tracking information received. Task τ_3: $C_3 = 30$; $T_3 = 160$;
- A periodic task responsible for reporting status across the network with a given computation time and period. Task τ_4: $C_4 = 10$; $T_4 = 300$;
- In addition there is an existing non-real-time application which requires 9% of the CPU cycles to meet its performance goals.

5.1.0.1. Partitioning of End-to-End Deadlines

When a message is sent within a station, it can be implemented by passing a message pointer to the receiving task and hence can be treated as any other OS overhead. However, when a message is sent outside the processor boundary, an integrated approach to assigning message and task deadlines needs to be developed. Consider the situation in Figure 1-1:

- The sensor takes an observation every 40.
- The signal processing task processes the signal, averages the result every 4 cycles, and sends it to the tracking processor every 160.
- The tracking processor task executes with a period of 160. It then sends a message to the control processor
- Task τ_3 on the control processor that uses the tracking information has a computational requirement of 30, and a period of 160 as given above. Recall that in order for the control processor to respond to a new observation by the sensor, the end-to-end latency needs to be less than 785.

A guiding principle in partitioning the deadline is to try and minimize the impact of workload changes in a subsystem and to contain the impact within the subsystem. If each resource is allowed a full period delay, each subsystem can be analyzed as if it is an independent resource. An alternate approach is to determine the completion time at each resource and the end-end delay is the sum of the completion times. This approach is more sensitive to workload changes.

Finally, when a task is scheduled on multiple resources in series, it may arrive at the next resource well before its deadline on the current resource. If we schedule the task immediately upon its arrival, it will create the *jitter* problem as illustrated below:

> Example 4: Consider two resources R_1 and R_2 connected in series. Assume task τ_1 has a period of 10. Furthermore, τ_1 is allocated a full period on each resource, and it uses each of the two resources for 5 units. Let task τ_2 use only the second resource for 3 units, with a period of 12 units. Let the first instance of τ_1 arrive at R_1 at t = 0, and let the first instance of τ_2 arrive at R_2 at t = 10.
>
> Suppose the first instance of τ_1 at resource R_1 completes its execution and arrives at

R_2 at t = 10. Since τ_1 has higher priority than that of τ_2, it will immediately use R_2, preempting τ_2. Observe that the second instance of τ_1 arrives at R_1 at t = 10. Suppose this instance is not delayed at R_1. Then at t = 15, the second instance of τ_1 will begin to use R_2, further preempting τ_2's starting time to t = 10. As a result τ_2 will miss its deadline at t = 12.

The jitter effect can be easily controlled by a simple rule: a task becomes ready to use a resource at the beginning of a new period. Using this rule in the above example, the second instance of τ_1 will be buffered and will become ready to use R_2 only at t = 20. In the following discussion we assume that this rule is enforced. It should be noted that jitter control is a special case of the phase control aspect of preemption control.

The steps involved in deadline partitioning are as follows: First we try to use the rate monotonic priority assignment. Since rate monotonic analysis guarantees end-of-period deadlines, we assume that the end-to-end delay is the sum of the period for each resource. Since the signal processor averages four cycles, each 40 units long, its delay is up to 160. Each of the other resources has a delay up to one period which is 160. That is, the total delay using rate monotonic scheduling is bound by 4*40 + 160 + 160 + 160 + 160 = 800. If it were less than the allowable delay then rate monotonic priority assignment could be used for all the resources.

However the specified maximum allowable latency is 785. Hence we may need to use deadline monotonic scheduling for at least some of the resources in the path. From a software engineering viewpoint, it is advisable to give a full period delay for global resources such as the bus or the network since their workload is more susceptible to frequent changes. Since there are two bus transfers involved we attempt to assign a full period to each. We also attempt to assign a full period to the signal and tracking processors. Hence the required completion time of the control processor task τ_3 should be no greater than $785 - 4 \times (160) = 145$. We therefore assign a deadline of 145 to control processor task τ_3.

Deadline Monotonic Scheduling

The deadline monotonic scheduling algorithm is a generalization of the rate monotonic scheduling algorithm. Rate monotonic scheduling assumes that the deadline of a periodic task is at the end of each period. That is, the period represents the window of time within which a task must initiate and complete its execution. Liu and Layland proved that it is optimal to give tasks with narrower windows higher priorities. They referred to this priority assignment method as the rate monotonic scheduling algorithm [2].

However, a task may have its deadline before the end of its period, resulting in a window narrower than its period. Leung and Whitehead proved that it is still optimal to assign higher priorities to tasks with narrower windows. They referred to this priority assignment method as the deadline monotonic scheduling algorithm [1].

References

1. Leung, J. and Whitehead, J. "On the complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks", *Performance Evaluation* (2), 1982.

2. Liu, C. L. and Layland J. W. "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment", *JACM*, 1973.

5.1.0.2. Scheduling Tasks on the Control Processor

We will concentrate on scheduling analysis of tasks in the control processor using the completion time test. Scheduling the backplane and other processors is similar. First, we need to create two virtual processors to separate the real-time and non-real-time applications. We select the length of the TDM cycle to be the same as the shortest period among the real-time tasks, 100, and the virtual processor switching overhead to be 0.5. Out of 100, 9 will be allocated to non-real-time activities and 90 to real-time virtual processor, with one unit lost in switching overhead.

Completion Time Test

The completion time test is a faster algorithm to test the schedulability of a task set than the scheduling point test that is usually used. It is based on the *critical zone* theorem [1] which states that if a task meets its first deadline even when all higher-priority tasks become ready at the same time then it can meet all future deadlines. Consider any task τ_n with a period T_n, deadline $D_n \leq T_n$, and computation C_n. Let tasks τ_1 to τ_{n-1} have higher priorities than τ_n. Note that at any time *t*, the total cumulative demand on CPU time by these *n* tasks is:

$$W_n(t) = \sum_{j=1}^{n} C_j \left\lceil \frac{t}{T_j} \right\rceil$$

The term $\lceil t/T_j \rceil$ represents the number of times task τ_j arrives in time *t* and therefore $C_j \lceil t/T_j \rceil$ represents its demand in time *t*. For example, let $T_1 = 10$, $C_1 = 5$ and $t = 9$. Task τ_1 demands 5 units of execution time. When $t = 11$, task τ_1 has arrived again and has a cumulative demand of 10 units of execution. Suppose that task τ_n completes its execution exactly at time *t* before its deadline D_n. This means that the total cumulative demand from the *n* tasks up to time *t*, $W_n(t)$, is exactly equal to *t*, that is, $W_n(t) = t$. A technique for finding this time is given in Figure 5-1.

Set $t_0 \leftarrow \sum_{j=1}^{i} C_j$

$t_1 \leftarrow W_i(t_0)$;

$t_2 \leftarrow W_i(t_1)$;

.

.

.

$t_k \leftarrow W_i(t_{k-1})$;

Stop when $(W_i(t_k) = t_k)$

Figure 5-1: Finding minimum *t*, where $W_i(t) = t$

Example 5: Consider a task set with the following independent periodic tasks:

- Task τ_1: $C_1 = 20$; $T_1 = 100$; $D_1 = 100$;
- Task τ_2: $C_2 = 90$; $T_2 = 145$; $D_2 = 145$;

Task τ_1 is clearly schedulable. The schedulability of τ_2 can be tested as follows:

$t_0 = C_1 + C_2 = 20 + 90 = 110$

$t_1 = W_2(t_0) = 2C_1 + C_2 = 40 + 90 = 130$

$t_2 = W_2(t_1) = 2C_1 + C_2 = 40 + 90 = 130 = t_1$ Hence task τ_2 finishes at 130 and is schedulable.

References

1. Liu, C. L. and Layland J. W. "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment", *JACM*, 1973.

Let the real-time task set in the control processor execute on the real-time virtual processor. The effect of the TDM cycle spent in non-real-time and overhead processing can be modeled as a high-priority task with a period of 100 and execution of 10. Consider the requirement for aperiodic event handling with an execution time of 5 and an average inter-arrival time of 100. We create an equivalent sporadic server task with 10 units execution and a duration of 100 which has the highest priority. A simple approximate analysis consists of two parts.

- First, the aperiodic arrives during the real-time virtual processor operation with 90% probability. Since we have allocated twice the average required bandwidth, we assume that the probability of an aperiodic arriving when there is no server capacity is negligibly small. Together with the fact that the aperiodic task has the highest priority, we can use a a simple M/D/1 queueing formula. We have the following result [3]:
- $W = \frac{\rho C}{2(1-\rho)} + C = 5.132$
- where $\rho = 5/100$, the utilization by the aperiodic task, and C = 5.
- Secondly, with 10% probability the aperiodic message arrives during the non-real-time virtual processor operation. Since the average aperiodic inter-arrival time is ten times longer than the duration of the non-real-time virtual processor, we assume that at most one aperiodic message can arrive when the virtual processor is executing. In this case the aperiodic message must wait, on average, for half the duration of the non-real-time processor including switching overhead. In this case the response time of the aperiodic message is $5+5.132=10.132$.

Finally, considering both cases, the response time of the aperiodic task is $0.9\times5.132+0.1\times10.132=5.632$. It is important to note that the analysis of aperiodic tasks is generally complex and may require simulation.

Since the TDM cycle and sporadic server have the same period, they may be considered as a single task; i.e., Task τ_1: C_1 = 20; T_1 = 100. Therefore, tasks on the control processor are as shown in Table 5-1:

Task	C	T	D
τ_1	20	100	100
τ_2	78	150	150
τ_3	30	160	145
τ_4	10	300	300

Table 5-1: Tasks on the Control Processor

Let tasks τ_1, τ_2, and τ_3 share several data structures guarded by semaphores. Suppose the duration of critical sections accessing shared data structures are bounded by 10 units. Suppose the priority ceiling protocol is used. Since the priority ceiling protocol is assumed to be implemented, higher-priority tasks are blocked at most once for 10 by lower-priority tasks.

We now check whether or not τ_3 completes within 145 under rate monotonic priority assignment. Under rate monotonic assignment, τ_1 and τ_2 have higher-priority than τ_3. Hence, the completion of τ_3 can be calculated using the completion time test as follows:

$$t_0 = C_1 + C_2 + C_3 = 20 + 78 + 30 = 128$$

$$t_1 = W_3(t_0) = 2C_1 + C_2 + C_3 = 40 + 78 + 30 = 148$$

$$W_3(t_1) = 2C_1 + C_2 + C_3 = 148 = t_1$$

Therefore the completion time of τ_3 is 148, which is later than the required completion time of 145. In order to meet the deadline of 145 imposed by the maximum allowable latency requirement of the previous section, we use the deadline monotonic priority assignment. This makes task τ_3's priority higher than task τ_2's priority, since τ_3 has the shorter deadline.

Under this priority assignment, the schedulability of each task can be checked as follows: Task τ_1 can be blocked by lower-priority tasks for 10 , i.e., B_1 = 10. The schedulability test for task τ_1 is a direct application of Theorem 1:

$$\frac{C_1}{T_1} + \frac{B_1}{T_1} = 0.2 + 0.1 = 0.3 \leq 1(2^{1/1} - 1) = 1.0$$

Therefore, task τ_1 is schedulable. Task τ_3 is the second highest priority task. Since τ_3 has a

deadline shorter than its period, the schedulability test for τ_3 can be checked as follows. Let $E_3 = (T_3 - D_3)$. In the schedulability test of τ_3, the utilization of task τ_2 does not appear, since τ_2 has a lower-priority and does not preempt τ_3. Because τ_2 has a lower priority, its critical section can delay τ_3 by 10 . Therefore $B_3 = 10$.

$$\frac{C_1}{T_1} + \frac{C_3}{T_3} + \frac{E_3}{T_3} + \frac{B_3}{T_3} = 0.2+0.188+0.094+0.0625 = 0.545 \leq 2(2^{1/2}-1) = 0.828$$

Now consider the third highest priority task τ_2. From the view point of the rate monotonic assignment, the deadline monotonic assignment is a "priority inversion". Therefore in the schedulability test for task τ_2, the effect of blocking has to include τ_3's execution time. The blocking time is $B_2 = C_3+0$. The zero indicates that there can be no lower-priority task blocking τ_2:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{B_2}{T_2} = 0.2+0.52+0.2 = 0.92 > 2(2^{1/2}-1) = 0.828$$

The schedulability test of Theorem 1 fails for τ_2. The schedulability of τ_4 can be checked by the following simple test since there is neither blocking or deadline before its end of period:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} + \frac{C_4}{T_4} = 0.2+0.52+0.188+0.033= 0.941 > 4(2^{1/4}-1) = 0.757$$

Note that the schedulability test of Theorem 1 fails for both tasks τ_2 and τ_4. To determine their schedulability we use the completion time test. Since τ_1 and τ_3 must execute at least once before τ_2 can begin executing, the completion time of τ_2 can be no less than 128:

$$t_0 = C_1 + C_2 + B_2 = 20 + 78 + 30 = 128$$

However, τ_1 is initiated one additional time in the interval (0,128). Taking this additional execution into consideration, $W_2(128) = 148$:

$$t_1 = W_2(t_0) = 2C_1 + C_2 + B_2 = 40 + 78 + 30 = 148$$

Finally, we find that $W_2(148)=148$ and thus the minimum time at which $W_2(t) = t$ is 148. This is the completion time for τ_2. Therefore τ_2 completes its first execution at time 148 and meets its deadline of 150:

$$W_2(t_1) = 2C_1 + C_2 + B_2 = 40 + 78 + 30 = 148 = t_1$$

Similarly we can check the schedulability of task τ_4 using the completion time test. It turns out to be schedulable.

5.2. Scheduling Messages on FDDI

The messages that exist on the FDDI network are as follows:

Station S_1: Sensor data is collected and stored. Every 150, the station transmits 1.0 Mbits of data. Let the station also transmit 5 Mbits of data every 100 to Stations S_2 and S_4.

Station 3: Video information: The required end-to-end deadline is assumed to be 100. As an example we assume a video source of 1024x768 pixels per frame with 24 bits/pixel at 30 frames/second, compressed with a ratio of 16:1. There also exist 3 channels of CD quality audio. Each channel is sampled at 44.1 Khz with 32 bits/sample. The end-to-end deadline for audio is also 100.

Consider scheduling of messages at Station 3. We need to partition the end-to-end deadlines into subsystem deadlines. The resources that need to be scheduled along the path between the source and the control processor are: the source interface processor, the network, the destination network interface, the backplane, and the control processor itself. As discussed in Chapter 5.1.0.1, the simplest way to partition the end-to-end deadline is to allow a delay up to a period on each resource.

First consider the video task. Its natural period at 30 frames/sec is 33. If we spend an entire period on each of the five resources, the end-to-end delay will exceed the limit of 100. Hence, we transform the sending period to 60 Hz, i.e., we send half a frame every 16.5. For the resolution given above this works out to no more than 6 of transmission time every period of 16.5.

Now consider the audio task. Its natural period is roughly one sample every 22 microseconds. This period is too short for the network as well as the packetization processing. Hence we transform the transmission period to 11; that is, we accumulate 500 samples every 11 units for each of the three sources. This bundling results in efficient network utilization, but requires the destination to buffer and regulate the delivery of the voice packets at the source frequency. This yields no more than 0.5 of transmission time every period. The end to end delay over 5 resources will be no more than 55.

Each source of traffic first has to packetize the information. The schedulability analysis of the tasks running on the source network interface processor is simpler than the analysis of the control processor tasks since there is no complex data-sharing between tasks. Hence we will omit the analysis and assume that it is schedulable.

Let the TTRT be 8 and let the walk time W_T be 1. The approach to scheduling traffic on FDDI is as follows. Consider the scheduling of messages in a particular station S_i with allotted synchronous bandwidth H_i. Therefore, the station can transmit up to H_i every TTRT. As discussed in Chapter 4, this can be treated as having another high-priority task with message transmission time (TTRT $- H_i$) and period TTRT. We refer to this task as Token Rotation task τ_{tr}. Using this framework, schedulability of traffic at each station can be independently analyzed.

Let us apply the above technique to messages at Station 3. Let H_3 = 4, then τ_{tr} = (C_{tr}=8–4=4, T_{tr} = 8). The message set for Station 3 is then:

- Token Rotation Message τ_{tr3}: C_{tr3} = 4; T_{tr3} = 8
- Audio Message τ_{13}: C_{13} = 0.5; T_{13} = 11
- Video Message τ_{23}: C_{23} = 6; T_{23} = 16.5

The schedulability of this message set can be checked using the completion time test. The token rotation message is obviously schedulable. Consider completion of message τ_1:

$t_0 = C_{tr3} + C_{13} = 4 + 0.5 = 4.5$

$t_1 = W_{13}(t_0) = C_{tr3} + C_{13} = 4 + 0.5 = 4.5$

Hence τ_{13} is schedulable. Consider completion of message τ_{23}:

$t_0 = C_{tr3} + C_{13} + C_{23} = 4 + 0.5 + 6 = 10.5$

$t_1 = W_{23}(t_0) = 2C_{tr3} + C_{13} = 8 + 0.5 + 6 = 14.5$

$t_2 = W_{23}(t_1) = 2C_{tr3} + 2C_{13} = 8 + 1.0 + 6 = 15.0$

$t_3 = W_{23}(t_2) = 2C_{tr3} + 2C_{13} = 8 + 1.0 + 6 = 15.0 = t_2$

Hence τ_{23} is schedulable.

Similarly, we can test the schedulability of messages at Station 1. If Station 1 is allotted a synchronous bandwidth of H_1 = 3, the message set at Station 1 can be written as:

- Token Rotation Message τ_{tr1}: C_{tr3} = 5; T_{tr3} = 8;
- τ_{11}: C_{11} = 10; T_{11} = 100;
- τ_{21}: C_{21} = 15; T_{21} = 150;

Note that this message set is also schedulable. The utilization of the network is 60 %.

6. Conclusion

The rate monotonic theory and its generalizations have been adopted by national high technology projects such as the Space Station, and have recently been supported by major open standards such as the IEEE Futurebus+. In this paper, we have given a synopsis of GRMS for centralized systems. Furthermore, we have described the basic concepts in the extensions of GRMS for distributed system scheduling, namely transmission schedulability, system scheduling consistency, and preemption control. We also introduced the notion of scheduling abstractions as a technique for analysis of systems that do not directly support GRMS. Finally, we have provided an application example to illustrate the assignment of message and task deadlines, task scheduling and message scheduling.

References

1. *IEEE 802.6 Distributed Queue Dual Bus - Metropolitan Area Network - Draft Standard - Version P802.6/D15.* October 1990.

2. Agrawal, G., Chen, B., Zhao, W., and Davari, S. "Guaranteeing Synchronous Message Deadlines in High Speed Token Ring Networks with Timed Token Protocol". *12th IEEE International Conference on Distributed Computing Systems* (1992).

3. Kleinrock, L. "Queueing Systems, Vol I". *John Wiley & Sons* (1975).

4. Liu, C. L. and Layland J. W. "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment". *JACM 20 (1)* (1973), 46 - 61.

5. Sha, Lui and Goodenough, John B. "Real-Time Scheduling Theory and Ada". *IEEE Computer* (Apr., 1990).

6. Sha, Lui, Rajkumar, R., and Lehoczky, John P. "Priority Inheritance Protocols: An Approach to Real-Time Synchronization". *IEEE Transaction On Computers* (September 1990).

7. Sha, Lui, Rajkumar, R., and Lehoczky, John P. "Real-Time Applications Using IEEE Futurebus+". *IEEE Micro* (June 1990).

8. Sha, Lui, Sathaye, S., and Strosnider Jay K. "Scheduling Real-Time Communication on Dual-Link Networks". *13th IEEE Real-Time Systems Symposium* (December 1992).

9. Sprunt, Brinkley, Sha, Lui, and Lehoczky, John P. "Aperiodic Task Scheduling for Hard Real-Time Systems". *The Journal of Real-Time Systems 1* (1989), 27-60.

10. Stankovic, J. "Real-Time Computing Systems: The Next Generation". *IEEE Tutorial on Hard Real-Time Systems* (1988).

11. van Tilborg, A. and Koob, G., . "Foundations of Real-Time Computing: Scheduling and Resource Management". *Kluwer Academic Publishers* (1991).