A DISTRIBUTED OPEN ENVIRONMENT FOR REAL-TIME APPLICATIONS

BY

LYNN Y. ZHANG

B.S., University of Illinois at Urbana-Champaign, 1997
M.S., University of Illinois at Urbana-Champaign, 1999

Dissertation

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Department of Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

Abstract

Traditional approaches to real-time schedulability analysis tend to require detailed timing attributes and resource usages of all applications that may run concurrently in the system. Reconfiguring such a system is expensive and often done offline. This fact has motivated the design of an open system architecture. Open system is a hierarchical reservation-based framework that provides temporal guarantee and isolation to independently developed applications. It is now possible to decide at runtime whether an application can meet its timing requirement based on a few overall parameters. However, open system was originally developed for applications to run on a single processor. This disseration generalizes the technique and extends the open architecture to a distributed environment. Many real-time applications are distributed in nature and require reservation support at the end-hosts as well as within the network. In a distributed system where a hybrid set of applications (i.e. hard, soft, and non real-time applications) may run concurrently in a cluster of processors connected together by a local area network, *"temporal isolation"* becomes a highly desirable property. The proposed distributed open architecture manages the CPU and network resources, and provides end-to-end timing guarantee to all real-time applications that are admitted into the system. The developer of each real-time application validates the schedulability of the application assuming it is running alone on a dedicated platform. If a real-time application is schedulable on a slower system of the required capacity and the open system admits the application, its temporal behavior will not be affected by any other applications in the system.

This dissertation validated the distributed open system for different types of networks: Myrinet, Controller Area Network (CAN), and Wireless LAN. While the hierarchical scheduling framework remains the same, several scheduling algorithms are developed to address the design issues

arise from the hardware constraints and/or industrial standards of each network. More specifically, Myrinet is a buffer-less wormhole switched network which requires special synchronization mechanism to avoid conflicts; CAN is a serial bus whose scheduling resolution is limited by the number of bits in the identifier of each message that can be used to encode priorities; Wireless LAN needs to conform to the IEEE 802.11 standard and consider one additional design parameter: energy efficiency, which is a trade-off between bandwidth utilization and the life span of wireless nodes. These issues influence the selection of algorithms used in different levels of schedulers in the hierarchical scheme. Simulation results show that real-time guarantee and inter-application isolation can be achieved with acceptable scheduling overhead. In some cases, the hierarchical scheduling scheme reduces the problem complexity and increases network utilization.

To my parents,

who made it possible.

To my husband and daughter,

who made it meaningful.

Acknowledgments

I am indebted to many people for their generous support during the long and sometimes difficult road to the completion of this dissertation. I wish to express my deepest gratitude to my co-advisers, Professor Lui Sha and Professor Jane Liu. Their guidance and advice had been extremely valuable in school and in life. Moreover, their dedication to hard work and excellence have been and will always be a constant source of inspiration for me. I would also like to thank other members of my committee, Professors Klara Nahrstedt and Jennifer Hou for their time, interest and support of this work.

Many past and present members of the Real-Time research group at University of Illinois have helped me in many different ways. I would like to thank Professor Marco Caccamo, Daniel Shih, and Zhong Deng for their encouragement and insightful discussions. I would like to thank Sathish Gopalakrishnan for his help in the final editing and formating of the dissertation.

I am extremely grateful to my parents, Liang-Cheng and Wenyu. Throughout the course of my study they have always been there for me. Without their care and encouragement, my completion of this dissertation would not have been possible.

I must also thank my husband, Tao, who has shared every bit of joy and hardship that I went through . I am indebted to his patience and many soul searching talks that helped me stay true to myself. Finally, I want to thank my daughter, Joy, for making the final achievement ever so sweet.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The dramatic performance increase in today's computing systems and network connectivity has led to an increasing desire to run distributed real-time applications (e.g. high-speed data acquisition, interactive multimedia application, factory automation, and decision support system) on a cluster of processors connected together by a high performance network. These real-time applications usually require bounded end-to-end delay. It is generally accepted that end-to-end delay guarantee in distributed system are provided by: (1) appropriate scheduling of resources at the end-hosts and within the network, (2) vigorous admission control mechanism, and (3) resource usage monitoring and enforcement in order to provide temporal isolation between applications [1].

These requirements have been well studied in the processor scheduling domain. Recently, an open system environment [1, 2] has been developed for applications that run on a single processor. The uniprocessor open system effectively provides a slower virtual processor for each of the real-time applications sharing the physical processor. Thus it makes possible the independent validation of the real-time performance of each application. The distributed open system architecture proposed in this dissertation extends the uniprocessor model with the following components:

- a communication server to perform network service on behalf of the applications,
- a hierarchical network scheduling scheme to provide timing guarantee and isolation to independently developed applications,

[1] A misbehaving application must not affect the temporal properties of other applications in the system.

- a simple admission mechanism to admit new real-time applications into the system at run-time, and
- scheduling algorithms suitable for each of the following networks: Myrinet, CAN, and WLAN.

1.1 Motivations and Objectives

Operating Systems providing resource reservation to support a hybrid set of workload, including hard, soft and non-real-time applications, have become an active research topic. However, in order to provide precise timing guarantee to real-time applications, vigorous schedulability analysis needs to be performed on the system. Oftentimes, the detailed timing attributes of all real-time applications must be known and the schedulability of every combination of applications that can run concurrently are determined *a priori*. Systems based on traditional approaches to real-time scheduling are closed in this sense. As each application is likely to be developed by different vendors, it's unrealistic to assume that the developers know about the other applications that will run concurrently in the system and follow a uniform scheduling algorithm. This fact led to the design of the open system architecture.

Many real-time applications are distributed in nature and require reservation support at the end-hosts as well as within the network. There exists two major challenges in extending the open system to a distributed environment: 1) co-scheduling the CPU and network resources, and 2) bounded network delay.

When multiple applications are running concurrently in a system, network access requires mutual exclusion. The kernel must determine which packet is to be sent at a given instant or which packets are discarded or processed upon reception; at the same time, it needs to decide when and for which application to process the protocol stack. Depending on the underlaying network, some specially tailored mechanisms are required to provide protection from misbehaving applications.

A wealth of bandwidth preserving scheduling algorithms is available in the literature includ-

ing the proportional generalized processor sharing [3, 4], virtual clocks [5] and etc. This class of algorithms are compatible with the hierarchical CPU scheduling scheme used in the open system and hence makes the network extension relatively simple. However, many of the existing distributed real-time systems are not connected together by ATM, or any other connection-oriented packet-switched networks. The Control Area Network (CAN), which is popular in industrial automation and robotics, is a typical real-time distributed system that cannot use any of the existing proportional-share or rate-based scheduling algorithms. We are hence motivated to design practical and backward compatible scheduling schemes for such networks. This dissertation presents a generalized network scheduling framework under which specific algorithms were developed for Myrinet, CAN, and IEEE 802.11 Wireless LAN.

In summary, the proposed distributed open system architecture should meet the following specific objectives:

- provide guarantee and timely access to CPU and network bandwidth for distributed real-time applications that are developed and validated independently,
  - support mixed scheduling policies: the developer is free to choose any scheduling algorithm suitable for the application,
  - enforce temporal isolation: once accepted into the system, each application has the illusion that it is running alone in a slower dedicated system,
- accommodate run-time reconfiguration,
  - avoid detailed global schedulability analysis: simple acceptance test based on a few overall application parameters,
- support heterogeneous applications in the system: hard, soft, and non-real-time applications,
- conform with existing network standards.

1.2 Contributions

The hierarchical scheduling scheme used in the open system for scheduling both the CPU and network resources provide application developers with the freedom to design and validate applications independently. In contrast to the traditional monolithic system where all tasks are scheduled together according to the same algorithm, each developer can choose the algorithms best suited to the application for scheduling CPU tasks as well as packet transmission. Timely and protected access to the CPU and network is provided between applications. This key property enables the independent development and run-time reconfiguration of distributed real-time systems.

According to the hierarchical scheduling scheme, each application is executed by a bandwidth-preserving server. There are three type of servers used for CPU scheduling in the open system: Total Bandwidth Server (TBS) [6], Constant Utilization Server (CUS) [1], and Passive Server (PS) [7]. The network bandwidth reserved by each application is each managed by a simple sporadic server[2] [8]. A lower level scheduler schedules these bandwidth-preserving servers on the EDF basis. The network scheduler has one additional level of scheduling hierarchy than the CPU scheduler because the network is a global resource to applications on different processors. We call this level the Medium Access Control (MAC) scheduler. To the MAC scheduler, each processor has a periodic reservation. The MAC scheduler at each processor is maintained by a simple sporadic server again which behaves exactly like a periodic message. The level of the hierarchy is useful in providing isolation between the processors, so that the applications will not suffer from a misbehaving application running on a different processor.

This dissertation studies three types of networks that are commonly used in distributed systems: Myrinet, Controller Area Network (CAN), and IEEE 802.11 based Wireless LAN (WLAN). Several schemes, each tailored to a specific network, are proposed to provide bounded delay and bandwidth isolation to real-time applications. Related works in different networks will be described with the proposed algorithms, as they will be presented in later sections. Furthermore,

[2] Aperiodic servers whose absolute deadline is computed according to the actual aperiodic traffic like the TBS and CUS are not suitable because they cannot be modeled like a periodic message.

this dissertation developed several heuristic algorithms for improving the performance of soft and non-real-time applications in CAN and WLAN networks.

1.3 Organization

The remainder of this dissertation is organized as follows.

Chapter 2 discuses related work. It presents three resource management schemes that have been proposed to enable inter-application resource isolation. This chapter also presents work related to co-schedule CPU and bandwidth to provide end-to-end timeliness.

Chapter 3 presents the distributed open system model. It describes the hierarchical scheduling scheme used in the operating system to create an illusion that each real-time application is running on a dedicated network that is a fraction of the physical bandwidth.

Chapter 4 addresses the problem of scheduling hard-real-time periodic messages in a wormhole switched network. It discusses how our distributed open system can be implemented with a two-level hierarchical scheduler on each PC system connected via Myrinet. It presents four algorithms to be used in the high level scheduler and one algorithm in the lower level scheduler. The chapter also explains how admission control can be done without detailed global scheduling analysis. Simulation results are also presented to show that hierarchical scheduling scheme outperforms the corresponding one level algorithm.

Chapter 5 addresses the problem of scheduling real-time messages, sporadic and aperiodic along with periodic messages on CAN bus. Traditionally CAN relies on bit-wise arbitration to establish transmission order. This chapter describes how hierarchical scheduling scheme can be applied here and presents the Implicit-EDF algorithm to be used in the lower level scheduler. The scheduling scheme achieves higher bandwidth utilization while avoiding the complexity of the deadline encoding problem.

Chapter 6 studies the scheduling in IEEE 802.11 wireless network. In addition to meeting the timing requirement, it also considers the energy constraints in wireless network. An energy

efficient real-time scheduling algorithm is proposed to minimize power consumed by the wireless transceivers of portable devices. A simulation study on the energy saving factor and overhead is presented.

Finally, section 7 concludes the dissertation with a summary of the results.

Chapter 2

Related Work

In this chapter, we describe the previous work that is related to the open system framework this dissertation is based on. The related work is in the following three areas: scheduling algorithms, resource reservation model, and real-time communication server extension. We will defer the related work on specific network till the Chapter 4, 5, and 6 which focuses on the scheduling scheme specific for each network.

2.1 Scheduling Algorithms

As we will describe in detail in the next Chapter, open system uses a two-level hierarchical scheme to schedule the applications in the system. Each real-time application is executed by a server. The low-level scheduler maintains the servers (i.e., provides them with execution budgets and sets their deadlines) and selects the server with the earliest deadline to execute. The high-level scheduler of each server uses an algorithm chosen for the application to schedule the ready jobs of the application. There are three types of servers used in the open system: constant utilization server [9], total bandwidth server [6], and passive server. The first two servers are used to execute regular applications and the last is used to execute service provider which will be discussed in detail in Chapter 4.

The constant utilization server algorithm is essentially the same as the total bandwidth server algorithm developed by Spuri and Buttazo [6]. The latter algorithm was originally proposed for

scheduling aperiodic jobs in the midst of periodic tasks in order to enhance the response times of the aperiodic jobs without affecting the schedulability of periodic tasks. A constant utilization server or a total bandwidth server is characterized by its size. The budget of a server is zero initially, and so is its deadline. The server budget is replenished for the first time when a job arrives to the ready queue of the server. When the budget of a server of size U is replenished to e time units at time t, the new server deadline is set to $\max\{t, d\} + e/U$, where d is the current server deadline. A server is ready for execution only when it has a budget, and it consumes its budget at the rate of one per unit time when it executes. According to the total bandwidth server algorithm, the server budget is replenished immediately after the budget becomes zero if the ready queue of the server is not empty. In contrast, according to the constant utilization server algorithm, the budget is replenished no earlier than the current server deadline.

The worst case response time is the same for a job executed by a constant utilization server or a total bandwidth server, while the job has a shorter average response time when executed by the total bandwidth server. This is because a constant utilization server does not make use of any background time, while a total bandwidth server does. We use constant utilization servers to execute hard-real-time applications in the open system whenever it is possible to do so and when there is no advantage to complete jobs in the applications early. Thus, we leave the processor time not required by such applications to non-real-time and soft-real-time applications. We use a total bandwidth server to execute all the non-real-time applications and soft real-time applications so that the system is more responsive for them.

There is a wealth of network bandwidth management schemes such as PGPS (proportional generalized process sharing) [4], WF^2Q (weighted fair-share fair-weighted queueing) [10], and virtual clock [5]. However, these algorithms were designed for switched network with buffer. Hence they cannot be directly applied to networks such as Myrinet (wormhole network) and CAN (bus network). Furthermore, the fluid-model that the above mentioned fair queueing algorithms were based on is too fine-grained and hence incur high run-time overhead. Our open system architecture allows end-to-end reservations by application which leads to a more general model for

scheduling messages on diverse network topologies.

Both CPU and bandwidth resources need to be managed jointly to provide end-to-end guarantee. The delay in sending and receiving messages needs to be included in the analysis. Related works in reservation-based design include the Mach μKernel [11] and its real-time extension RT-Mach [12] which enables the correct accounting of execution cost of tasks. Mercer, et al. [13] proposed the processor reserve abstraction in RT-Mach to enforce proper resource usage. Rajkumar, et. al. extended the processor reserve abstraction to a uniform model that includes all time-multiplexed resources. This model forms the basis of Resource Kernels [14, 15].

A communication link with link bandwidth C can be thought of as a processor, and a total bandwidth server with server size U can be thought of as a real-time connection that is allocated the bandwidth U_C and scheduled according to the virtual clock algorithm. Indeed, the two-level hierarchical scheduling algorithm for nonpreemptive applications can be used to schedule multiple real-time message streams on each of the connections sharing an output link of an ATM switch.

The two-level scheduling scheme used in the open system resembles the proportional share resource allocation scheme proposed by Stoica et al. Their scheme provides fair sharing of a processor among multiple processes running on the processor, but does not guarantee their timely completion. In contrast, our scheme guarantees that the deadlines of multi-threaded, hard-real-time applications on a processor are always met, but provides fair sharing of the processor only to non-real-time applications. In particular, our scheme allows different applications to be scheduled according to different scheduling algorithms and the schedulability of each real-time application to be validated independently of other applications.

We will show in the following chapters that the hierarchical scheduling algorithm can be used in other networks such Myrinet, CAN, and Wireless LAN if we are able to select the algorithm for the lower and/or higher level server wisely.

2.2 Resource Reservation

Both CPU and bandwidth resources need to be managed jointly to provide end-to-end guarantee. The delay in sending and receiving messages needs to be included in the analysis. Related works in reservation-based design include the Mach μKernel [11] and its real-time extension RT-Mach [12] which enables the correct accounting of execution cost of tasks. Mercer, et al. [13] proposed the processor reserve abstraction in RT-Mach to enforce proper resource usage. Rajkumar, et. al. extended the processor reserve abstraction to a uniform model that includes all time-multiplexed resources. In a system based on process reserve model, the operating system maintains and enforces a processor reserve for each real-time task in the system. A processor reserve is defined by a 2-tuple (σ, T) of processor utilization σ and reservation period T (i.e., it has budget σT). For any task having processor reserve (σ, T), the system guarantees that during each period T, the job of the task receives at least σT processor time.

Any real-time task that is to run in the system must specify its processor reserve. If the task is periodic, the reservation period T is equal to its period. If the task is aperiodic, the reservation period T is determined by the delay requirement of the task. The system subjects each request to an acceptance test according to the scheduling algorithm used by the system. When a job of an accepted task executes, the time it uses is subtracted from the budget specified in the reserve. If the budget reaches zero but the job does not complete, the job loses its real-time priority and can only execute by using background time in a time-sharing fashion. At the end of each reservation period, the reserve gets a new allocation of the budget which can be consumed during the subsequent reservation period.

In our open system, service providers, such as files servers and network protocol stack handlers, use a similar approach. An application requesting a service gives the service provider its own execution budget so that the provider can execute on its behalf. The execution budget in our open system, specified by a 2-tuple (budget, deadline), is analogous to Mercer's reserve abstraction. One difference is that our execution budget is not replenished periodically; once it is consumed, the

execution budget no longer exists. The application must give the service provider a new execution budget if it needs more service. Mercer uses processor reserves as a means to monitor and control the processor usage of each activity. The budget replenishment scheme used in the open system has a similar purpose. It is designed to prevent each application from overruning its reservation. In addition, it allows the service provider to deliver the requested service in time and the requesting application to meet its deadline.

The concept of processor reservation is used in Resource Kernels [14] developed by Rajkumar. Similar to our open system, the Resource Kernels system supports multiple real-time applications and allows the start and termination of applications at run time. A real-time application running in such a system needs to specify the resource requirement of each of its threads in the form of processor reserves. The admission control is based on the processor reservation model using the resource requirement specified by the requesting applications. Unlike the open system, the the Resource Kernels system requires that all real-time applications be scheduled according to a fixed-priority scheduling algorithm (i.e., either RM or DM), and it uses a global schedulability analysis to conduct the acceptance test. The test uses the resource reservations of the requesting application, as well as those of the existing applications.

Mok [16] proposed a similar but more generic hierarchical real-time resource model that permits resource partitioning to be extended to multiple levels. Partitioning on each level are scheduled as if they had access to a dedicated resource. This model works well when there's a clean separation between task scheduler and resource scheduler.

Chapter 3

System Model and Proposed Approaches

A hierarchical scheme for scheduling network traffic is proposed to extend the open system environment to a distributed architecture by creating a virtual slower network for each application. Figure 3.1 illustrate the high level concept of an open system. In this figure, a cluster of PCs are connected by a high performance network. Two real-time applications are running concurrently in this cluster. The rectangle slices in the PC represent the minimum CPU resources each application needs in order to schedule its job, we call this the required processor capacity of the real-time application. Similarly, the lines connected the rectangles to the network show the minimum network bandwidth in which the application required to transfer all its message streams, we call this the required network capacity. The open system guarantees that if the real-time messages of any application is schedulable on a slower network of the required capacity and the open system admits the application, they will be schedulable in our distributed open environment.

3.1 Hierarchical Network Scheduling Scheme

Figure 3.2 shows our model of distributed open system. Multiple distributed applications run on a cluster of PCs or workstations. To validate that an application can meet its timing requirements, the developer analyzes the schedulability of the application assuming that the application runs alone on dedicated processors and network whose speed and bandwidth are fractions of that of the physical processors and network. As mentioned previously, the minimum processor speed of

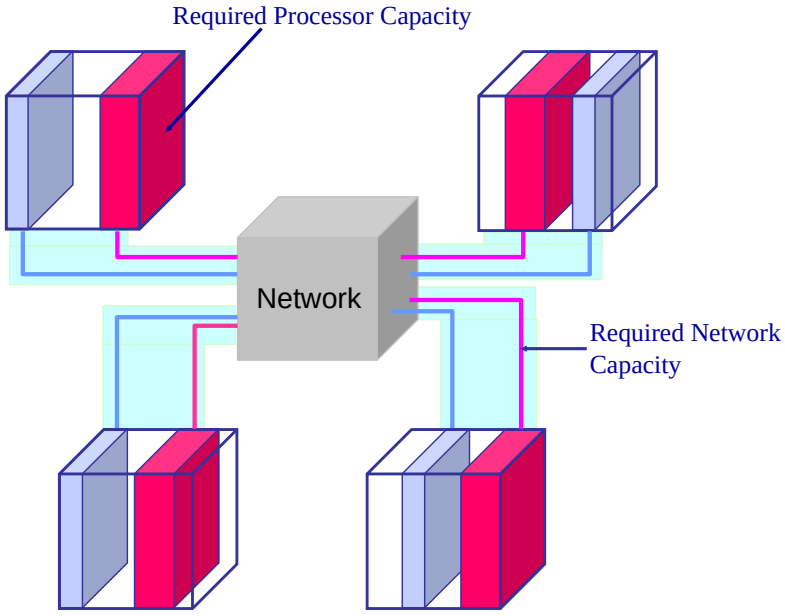


Figure 3.1: High level distributed open system model

σ_k and network capacity B_k at which application A_k is schedulable are the required processor and network capacities of the application. We normalize the required capacities of the applications with respect to physical processor speed and total network capacity, respectively. For example, the normalized required network capacity U_k of application A_k is B_k/U_{max} , where U_{max} is the total capacity of the target network on which the application will run and B_k is the minimum required network capacity of application A_k. For the sake of simplicity and without loss of generality, our discussion below assumes that U_{max} = 1 and the speed of the physical processors is also 1.

Each application, denoted by $A_k : k = 1, 2, \ldots, N$, generates multiple message streams. (The number N of applications may vary during runtime as some applications terminate and new ones start.) Each message stream is to be sent across the network to a receiving application. A periodic message M_i from A_k is characterized by an ordered tuple (src_i, dst_i, c_i, p_i), $i = 1, 2, \ldots, M$. src_i and

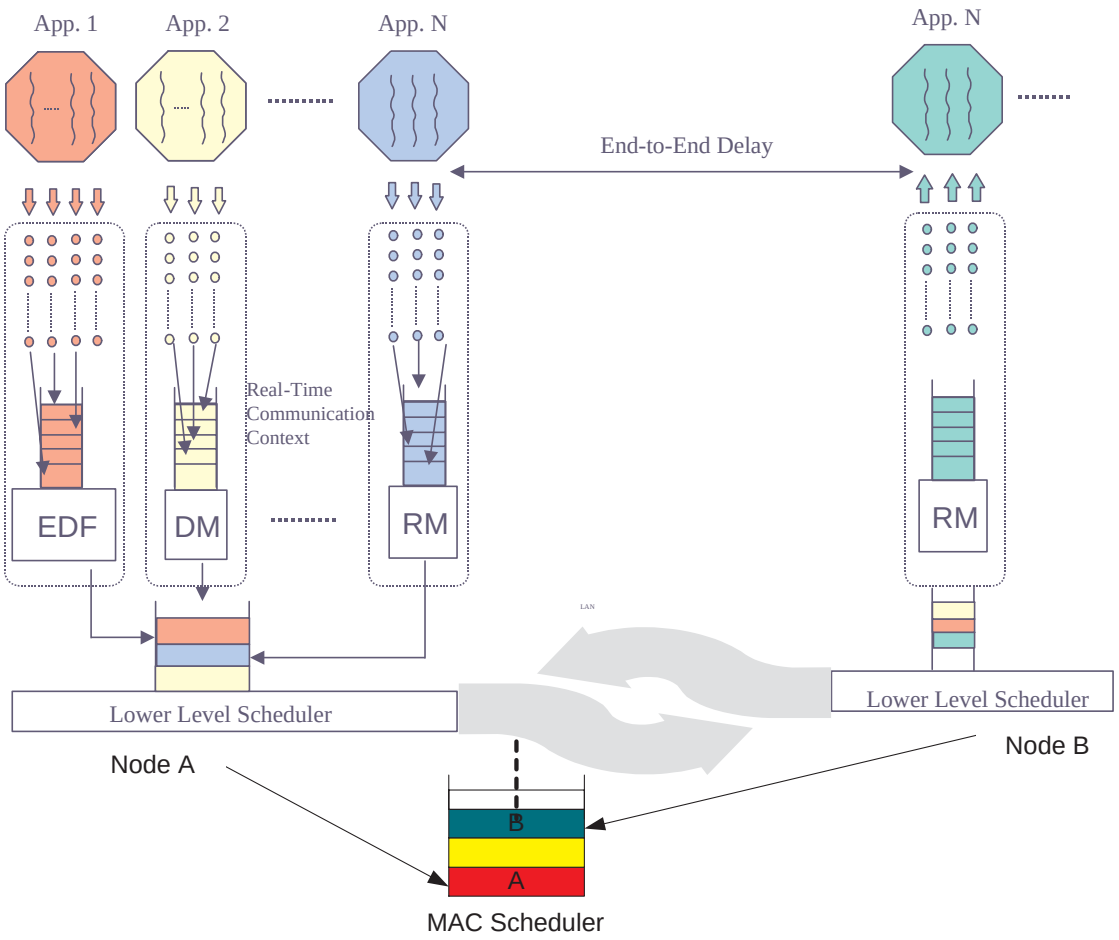


Figure 3.2: Distributed open system model

dst_i are the source and destination application ID of the message stream M_i, respectively; c_i is the number of packets generated per period; and p_i is the period of message stream M_i from A_k and assume that it is the relative deadline of the message stream[1]. It is assumed that the destination application ID dst_i will be mapped to a destination node ID in the lower level of the protocol stack.

The utilization of a periodic message M_i is $u_i = c_i/p_i$. The utilization U_k is the sum of utilizations of all messages belongs to A_k. Application A_k makes a reservation request of c_k slots in every p_k slots. The value p_k has to be less or equal to the smallest p_i for all message M_i in A_k. The effective utilization of A_k has to be rounded to $U'_k = \lceil U_k \cdot p_k \rceil / p_k$. The utilization U_j of node n_j is in turn

[1]Aperiodic messages can be represented by the same tuple where p_i is, in this case, the minimum inter-arrival time between any two messages.

the sum of the U_k''s of all the applications A_k running on n_j. We call U_j the node utilization of n_j. As shown in Figure 3.2, each node in the network needs to be scheduled for medium access. Each node n_j then makes a reservation of c_j slots in every p_j slots. Again, the node utilization needs to be adjusted to the p_j value in order to avoid reservation a fraction of the slot, $U_j' = \lceil U_j \cdot p_j \rceil / p_j$. Figure 3.3 illustrate the hierarchy of reserve abstractions.

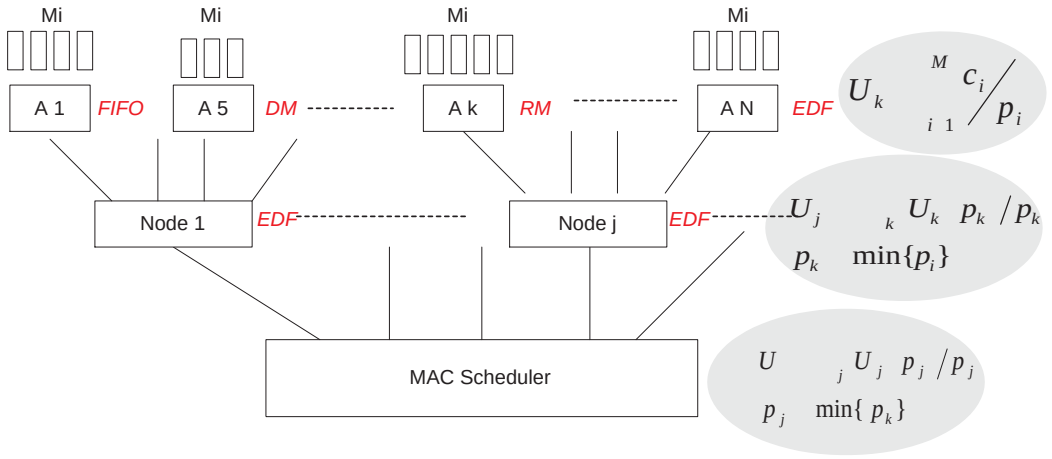


Figure 3.3: Reservation hierarchy in the open system

At initiation time, all applications run in the non-real-time mode. The operating system serves non-real-time applications on the best effort basis. When a distributed real-time application requests admission into the open system (i.e. to execute in real-time mode), it specifies in its request its required network capacity as well as its required processor speed. If after granting the request, 1) the application passes the processor acceptance test[2], and 2) it passes the schedulability test at each level of schedulers in the scheduling hierarchy, then the system accepts the application.

Each task in the application consists of a chain of CPU jobs and message transmission jobs. On each processor, a two-level CPU scheduling scheme makes it possible for the operating system to guarantee the timing constraints of CPU jobs of individual applications while allowing the CPU jobs of each application to be scheduled according to the algorithm chosen for the application. When a job requests networking services, it passes its CPU execution budget to a communication server in the system. The CPU time consumed by the communication server to send and receive

[2]Detailed presentation of the acceptance test on uniprocessor open system can be found in [1].

messages on behalf of the job is thus charged to the application to which the job belongs. The communication server schedules all outgoing and incoming messages and guarantees the timely delivery of each message, while at the same time provides bandwidth isolation to all real-time applications in the system.

The algorithm used to schedule out-bound messages is also a hierarchical scheme. The message streams from every application are scheduled by a high level scheduler (*HLS*). Specifically, all messages from non-real-time applications are scheduled in FIFO order by the scheduler which we call HLS_0. (This is how messages are scheduled in general purpose operating systems.) Messages from each real-time application A_k , for $k > 0$, are scheduled by a scheduler HLS_k using an algorithm chosen by the developer of the application . Each high level scheduler has a ready queue to hold out-bound messages of application A_k . A low level scheduler (LLS_j) multiplex the packets from all the HLS_k on node j onto the network whenever node j has permission to transmit. The node transmission order is determined by a MAC scheduler. The scheduling algorithms used by each of the schedulers vary according to the underlying network, which will be described in details in the later part of the disseration.

3.2 Network Subsystem

As described earlier, the communication server is an important component in providing end-to-end guarantees in the distributed open system design. It is responsible for conducting the network acceptance test, implementing the hierarchical network scheduler, and controlling the interaction between CPU and the network. The communication server is implemented as a user-level application that execute on a dedicated *passive server* which is designed to support system service applications. To keep accurate accounting of the CPU resource, the communication server is given only a very small amount of processor capacity and uses its own execution budget solely for administrative purposes. The processor time the communication server uses to perform any service is charged to the application requesting the service.

A real-time client application requests a service by calling one of two real-time application programming interface (RTAPI) calls, *SendData* and *RecvData*. When an application A_k calls the function *SendData*, it passes in the request the data to be sent and the handle to the server, as well as its own execution budget and deadline for consuming the budget. In response to the request, the communication server creates a new work thread in its own address space. The work thread inherits this budget and deadline. This thread is then inserted into the ready queue of the passive server on EDF basis. When the work thread is scheduled to run, it copies data into the communication context that has been reserved for the application. Also, it schedules the packets in the application's message queue according to the algorithm used by the application. The lower level scheduler can be implemented as a control program on the network interface level or as a periodic task of the communication server. It schedules all the high-level schedulers using the EDF algorithm. When a processor has permission to send to the network, the low-level scheduler select the message at the head of the message queue from the active scheduler with the highest priority.

On the receiving host, a client application calls RecvData to request the communication server to process incoming data on its behalf. Similar to SendData, the RecvData function also causes a transfer of budget from the requesting application to the communication server. If the data to be received is not available when the requesting application calls *RecvData*, the call effectively freezes the requesting application: the server of the requesting application has no budget, and the communication server holding the application's budget cannot execute on behalf of the application as the data is not available. To circumvent this problem, the application first makes a call to WaitData to indicate it is expecting incoming data. If the data is available, WaitData returns immediately. Otherwise, the CPU thread calling WaitData is blocked. Since WaitData function does not transfer budget to the communication server, other threads in the requesting application may still execute during the time when the calling job is blocked. When incoming data arrives, the communication server processes the message header and then notifies the receiving application about the source and length of the message. The application is expected to respond immediately. As soon as the thread calling RecvData is scheduled by the CPU scheduler and the application server is replen-

ished, execution budget is transferred from the application to the communication server to process the packets and copy the message from the communication context to the application's address space.

To summarize this section, we have presented the scheduling hierarchy and described the operation of the communication server at the end-hosts. The CPU scheduling scheme used in the open system guarantees timely completion of sending and receiving CPU jobs. In the next three sections, we present the algorithms used to schedule real-time messages in Myrinet, CAN, and IEEE 802.11 Wireless LAN, respectively.

Chapter 4

Scheduling on Myrinet

Myrinet [17] is a gigabit-per-second switch network which employs worm-hole routing. The core of the switch is a pipelined crossbar (i.e. buffer-less switch). The network interface has a programmable processor, called LANai, which has 1024K SRAM for storing the Myrinet Control Program (MCP) and for packet buffering. The network interface is capable of direct interaction with host processes and achieves a peak application-to-application bandwidth of 650Mbps and latency as small as 11 μs.

With the availability of high bandwidth and low latency interconnect, a cluster processors connected together by Myrinet can now support applications that used to require a supercomputer. The advantage of cluster computing which include lower cost and higher availability has led to a flurry of cluster research in the last ten years. In particular, there is an increasing desire to run high performance real-time applications (e.g. high-speed data acquisition and real-time simulation) on a cluster of inexpensive PCs connected by a high speed network. The typical scenario is that multiple hard-real-time applications are running concurrently in the Myrinet cluster. Each application is executing on a number of processors and a large volume of data needs to be changed between these processors.

Each application developer has *a priori* knowledge of the timing information of all real-time messages that will be transmitted by the application at each processor. The developer design and validated the application based on the assumption that the application is running alone on dedicated processors and network whose speed and bandwidth are factions of that of the physical processors

and network.

Assuming time is divided into fixed size time-slot. During each slot, the switch can receive at most one packet from each input link and transmit at most one packet on each output link. Whenever the header of a packet reaches the switch, if the output link is not being used for the transmission of another packet, the remaining portion of the packet is forwarded to the destination as soon as the header is decoded. If the output link is busy, the packet is said to be blocked. A simple STOP/GO flow control mechanism blocks the source from sending more bits into the link. If a packet is blocked for more than 50 *ms*, the packet is dropped. Hence, in order to provide deterministic timing guarantee to the real-time applications, our network scheduling scheme has to construct a conflict-free schedule.

4.1 Scheduling Periodic Messages over Myrinet

Several algorithms have been proposed for scheduling real-time messages through switches with buffer spaces. Well known examples are Weighted Fair Queueing (WFQ) [3, 4], Virtual Clock [5], Weighted-Round-Robin (WRR) [18], and Framed-Round-Robin (FRR) [19] algorithms. Our two-level scheme uses the WRR algorithm at the lower level to partition the network among applications.

Past work on scheduling periodic messages through a buffer-less switch [20, 21, 22, 23] have focused on finding transmission schedule for data traffic with no deadlines or isochronous traffic that has the same period, and hence, naturally fits into a single frame length. The objective is to assign packets to slots thats minimize schedule length and maximize link utilization.

However, in our scheduling problem, the period of the real-time messages are not the same. Each message has to be transmitted before the end of its period. The first intuition is to sort the messages using the earliest deadline first (EDF) or minimum laxity first (MLF) algorithm. But the schedule needs to be adjusted to avoid conflict. Therefore, an order should be defined, otherwise removing one conflict may cause another conflicts at a different link. It is obvious that the prob-

lem complexity grows enormously as the switch dimensions and number of messages increase. If we attempt to schedule all messages from all the applications together, performing an exhaustive search for conflict free schedule quickly becomes impractical. And it becomes impossible to provide any form of isolation between the applications. One misbehaving application can block the entire network.

The hierarchical network scheme comes in handy in this scheduling problem. The lower level scheduler (*LLS*) uses the weighted-round-robin algorithm to partition the network among applications. From the application scenario we described earlier, each application runs concurrently on multiple processors and makes aggregate bandwidth reservation. There is no need to distinguish multiplex applications from different processors onto the network. Hence the MAC scheduler is not needed in this case.

Each high level scheduler (*HLS*) uses one of four heuristic algorithms to construct conflict free schedule for the messages that belongs to each application. They are Earliest Deadline First Row-by-Row (EDF-RR), Minimum Laxity First Row-by-Row (MLF-RR), Earliest Deadline First using System of Distinct Representatives (EDF-SDR), and Minimum Laxity First using System of Distinct Representatives (MLF-SDR) algorithms.

All the algorithms represent the traffic in the system by a traffic matrix T . The *x*,*y*-th element $T_{x,y}$ of the traffic matrix contains a descriptor of each periodic message whose source is processor x and destination is processor y.

The row-by-row algorithms consider the traffic specified by each row of the traffic matrix, one row at a time. In other word, such an algorithm schedules the message stream from each source (i.e. one each input link) in turn, in order as the sources appear in the traffic matrix.

According to the EDF-RR algorithm, message streams from each source are scheduled on the EDF basis whenever possible. After message messages from sources $1, 2, \ldots x-1$ are scheduled, the scheduler then schedules the message streams from source *x* given by the *x*-th row of the traffic matrix. It does so by choosing a packet to be transmitted one slot at a time. For each time slot, it chooses the packet which has the earliest deadline among all ready but not yet scheduled packets.

Because of the possibility of conflicts, except the first row, the packet chosen for a time slot may not be one with the earliest deadline among all ready packets, and some slots may be left idle despite some packets being ready.

The MLF-RR algorithm is similar to the EDF-RR algorithm except the MLF rule is used to select the packet. Laxity of a message instance is equal to its deadline minus the current time and minus the time required to send the remaining packets in the message instance.

In contrast to the row-by-row algorithm, the SDR algorithms look at all outgoing messages specified by the traffic matrix before making any scheduling decision. The EDF-SDR algorithm first sorts all message instances given by the traffic matrix according to their deadlines. Let $d_1 \geq d_2 \geq \cdots \geq d_m$ be a list of distinct deadlines of all message instances that are to be transmitted during a hyper-period. During each time slot, the algorithm schedules as many packets as possible with the deadline d_1 , then d_2 , and so on in order of increasing deadline d_i, until $i = m$ or the slots on all input and output links are taken which occurs sooner. The MLF-SDR algorithm is similar to EDF-SDR, except that at each time slot the packets of message instances with the minimum laxity are chosen instead of those with the earliest deadline.

The application developer selects one of the four algorithms and validates that a feasible schedule can be constructed if the application is running alone in a slower virtual network[1] and all messages are scheduled by a one-level scheduler using the selected heuristic algorithm. After the required processor and network capacities are determined, the developer can initialize the real-time application and then ask the operating system to admit the application into the open system.

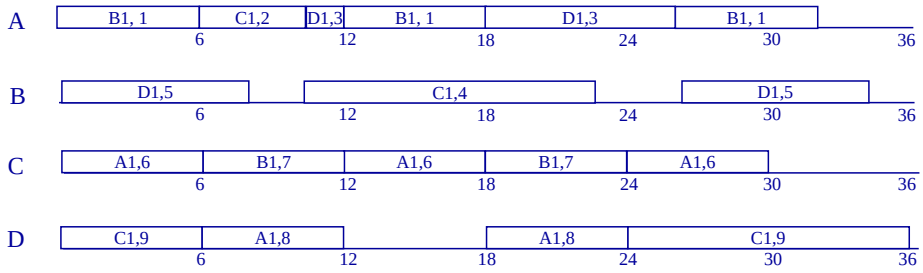
4.1.1 Illustrated Example

To better illustrate the hierarchical scheduling scheme, we consider an example where two distributed applications run together on processors A, B, C, and D connected via Myrinet. The tables in Figure 4.1 and 4.2 list the parameters of periodic messages from applications 1 and 2 respec-

[1]This implies that the time it takes to transmit a packet is proportionally longer compared to the faster physical network.

destination / source	A	B	C	D
A		($B_{1,1}$, 6, 12)	($C_{1,2}$, 8, 36)	($D_{1,3}$, 8, 18)
B			($C_{1,5}$, 12, 36)	($D_{1,4}$, 8, 18)
C	($A_{1,6}$, 18, 36)	($B_{1,7}$, 6, 18)		
D	($A_{1,8}$, 6, 18)		($C_{1,9}$, 18, 36)	

Application 1's Message Characteristics

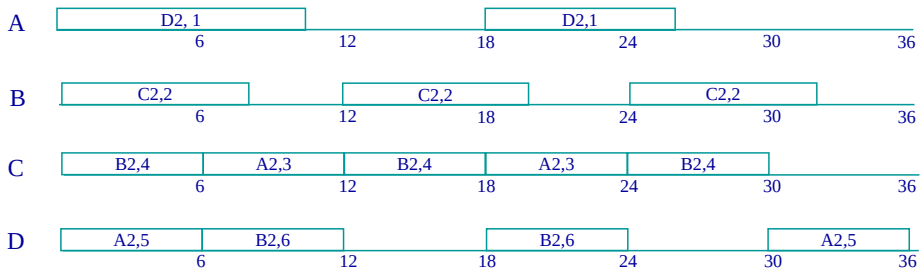


Conflict-Free Schedule for Application 1 constructed with EDF-RR

Figure 4.1: Conflict-free schedule for application 1 using the EDF-RR algorithm

destination / source	A	B	C	D
A				($D_{2,1}$, 10, 18)
B			($C_{2,2}$, 8, 12)	
C	($A_{2,3}$, 6, 18)	($B_{2,4}$, 6, 12)		
D	($A_{2,5}$, 6, 18)	($B_{2,4}$, 6, 18)		

Application 2's Message Characteristics



Conflict-Free Schedule for Application 2 constructed with EDF-RR

Figure 4.2: Conflict-free Schedule for application 2 using the EDF-RR algorithm

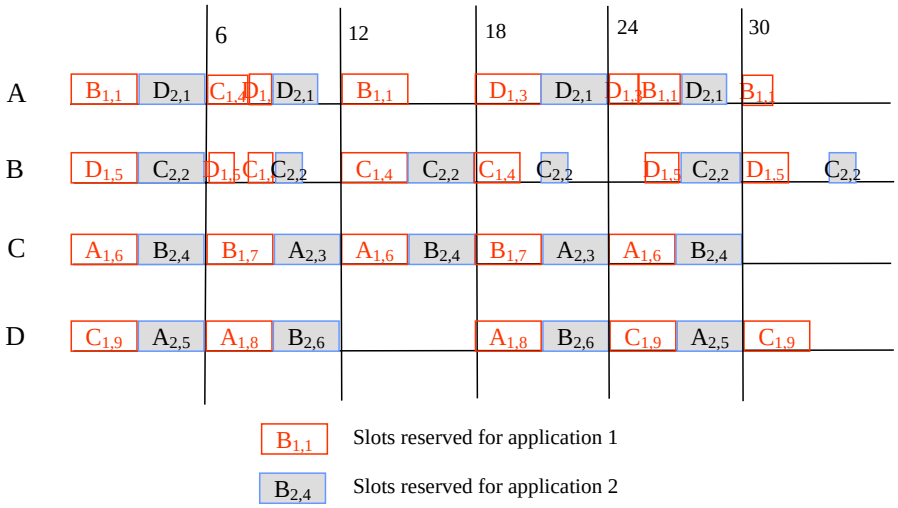


Figure 4.3: Application 1 and 2 running together in open system

tively. (We omit the name of the message source in the tuple denoting each message since the source is clear in the tables. We also number messages from both applications in sequence so we do not have to use double in the figure.) The relative deadline of every message is equal to its period. Figures 4.1 and 4.2 show conflict-free feasible schedules for the message streams of applications 1 and 2 if each were to run alone on a dedicated system in which the network is a Myrinet whose network capacity is only half of the physical network. The conflict-free schedule for each application is constructed by a high level scheduler *HLS* using the earliest deadline first row-by-row algorithm. While the developer of a real-time application is free to choose any algorithm for the *HLS* based on the traffic characteristics of the application, the low level scheduler *LLS* is responsible for partitioning the physical network based on the applications' requested capacities. The LLS uses the weighted-round-robin algorithm to assign slots to each application. Application 1 and 2 in this example each reserves 50% of the total network capacity. The round length is set to 6, which is the minimal message inter-arrival time in the system. Each application is allowed to send 3 packets in every 6 time slots. The LLS transmits packets from HLS_1's ready queue during slot 1, 2, and 3 in each round, then switches to HLS_2's ready queue in slots 4, 5 and 6. Figure 4.3 shows a feasible schedule for application 1 and 2 running together in a fast network. The time it takes to transmit each message reduces in proportion with the the increase of network bandwidth, i.e (A_1, B_1 , 6, 12) (i.e. (B_1, 6, 12) in the table in Figure 4.1 is now (A_1, B_1, 3, 12). We can see all the periodic message streams meet their deadlines.

4.2 Admission Test

The open system subjects each new real-time application A_k to an acceptance test. In addition to the set of parameters required for the acceptance test on each processor, the application also provides in its request its required network capacity, the message scheduling algorithm Σ_k, and the minimum periods σ_k of all its message streams.

Step 1: The CPU acceptance test described in [7] must be conducted on the application to guarantee that the application is schedulable on each processor.

Step 2: For the network acceptance test, recall that the *LLS* partitions the network on weighted-round-robin basis. Whenever a new real-time application A_k starts to execute in the real-time mode, the *LLS* updates the length of the round C to the shortest message inter-arrival time in the system, i.e. $C = \min_j\{\delta_j\}, j = 1, ..., k$. δ_k is the minimum period of all message streams in A_k.

Step 3: However, because the messages are preemptable only at the boundary of each slot, the operating system needs to round up the requested capacity based on the round length, i.e. $U_j' = \lceil U_j \cdot C \rceil / C$, of A_j, for $j = 1, ..., k$. This modified value is called the effective required capacity.

Step 4: A_k passes the network acceptance test if after granting the effective required network capacity to A_k, $\sum_{j=1}^{k} U_j\prime \leq 1$, else reject A_k.

After the application passes the acceptance test, the operating system in each of the participating processors creates a high level scheduler. Again, the HLS_k uses the heuristic algorithm selected by the application developer of A_k. The lower level Myrinet scheduler updates its send schedule so that each application is given its reserved fraction of time slots in each round. The slots that are not assigned to real-time applications are used for non-real-time traffic, i.e. messages in HLS_0 's ready queue.

4.3 Simulation Study

We conducted a simulation study to evaluate the hierarchical scheduling scheme. For this purpose, we simulate two systems, a closed system and an open system. The systems consist of identical sets of real-time applications. We randomly generate message streams (src_i,dst_i, c_i, p_i) for each real-time application such that the input and output link utilization are less than or equal to the application's required capacity. We choose the required capacities for the real-time applications in

order to simulate different scenarios: (1) network bandwidths are equally divided among multiple applications; (2) network bandwidths are divided among applications with a range of bandwidth requirement; (3) one application reserves a large fraction of the network bandwidth while the rest share the remaining bandwidth (e.g. one application is flooding the network while the rest may be simple control programs that pass small packets between processors in the cluster).

The schedulability of both an open system and a closed system depend on the parameters of the system and the characteristics of the real-time applications running in the system. These parameters include the number of applications in the system, the number of message streams in each application, variation of message inter-arrival times, scheduling algorithm, utilization of each link, and the overall system load distribution. To study how each of the parameters of the system and the real-time applications affects the overall schedulability, we conducted a number of experiments. In each experiment, we generated a set of synthetic workload by randomizing one particular parameter while keeping the others fixed. Every real-time application in our study contains only periodic message streams.

For each experiment, the message streams of all applications in a closed system are scheduled together according to a heuristic algorithm. In an open system, we scheduled the message streams from each application according to the same algorithm as if it executes independently in a virtual slower network. We then merged the schedules as described in the previous section. Since we cannot mix the scheduling algorithms used by different applications in the closed system, we let all applications in the open system use the same scheduling algorithm as well. In both the closed and our open system, we applied one of the heuristic algorithms described earlier in an attempt to find a conflict-free schedule. We then count how often each algorithm found conflict-free feasible schedules in each system. This gives the relative success rate of each heuristic algorithm in both systems.

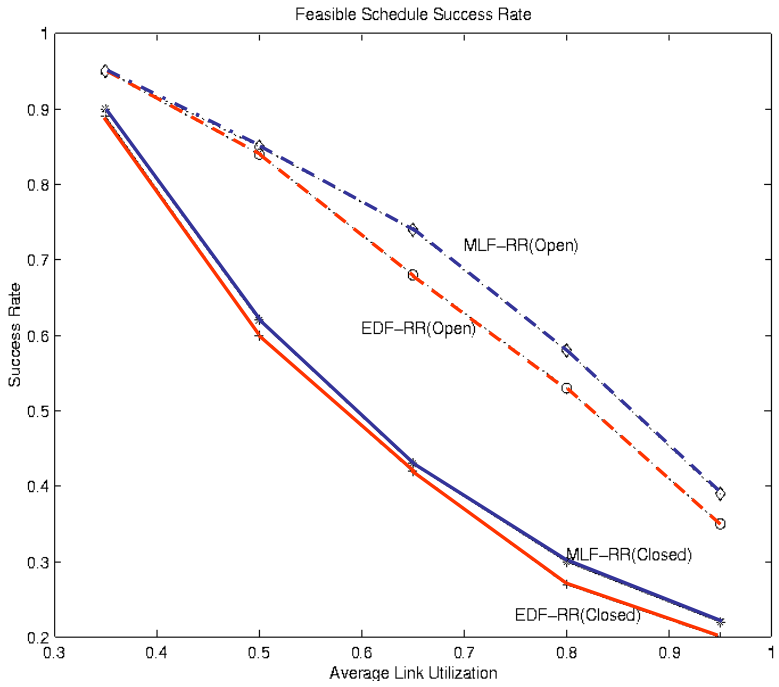


Figure 4.4: Schedule Length = 420, Switch Size = 4, Average Link Utilization = 90%

4.3.1 Baseline

The solid lines in Figure 4.4 and 4.5 give the performance for the four heuristic algorithms described above in closed systems. We generate 10000 message sets to yield negligibly small confidence intervals. Periods are selected so the schedule length (i.e. the least common multiplier of the periods of all messages) in each set does not exceed 420. We keep the average link utilization at 90%. The network is a 4-port Myrinet. We see that the SDR-based algorithms outperform the RR-based. MLF algorithm does a little better than the EDF algorithm. This is expected because MLF algorithm considers the number of packet remaining to be sent in addition to the deadline. Hence laxity is a better measure of urgency than deadline alone. SDR-based algorithms are more complex, but the performance gain justifies the complexity. (We expect they would outperform

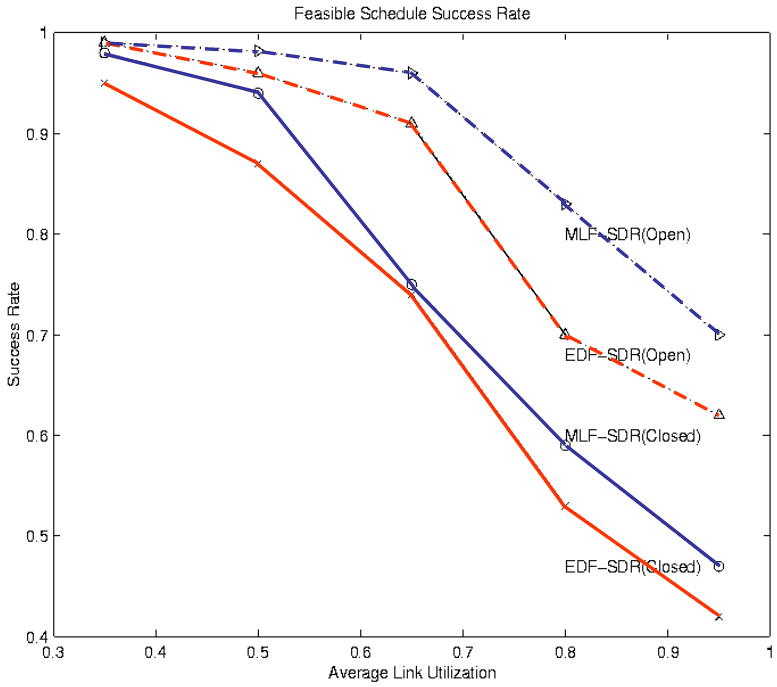


Figure 4.5: Schedule Length = 420, Switch Size = 4, Average Link Utilization = 90%

the Row-by-Row based algorithms, because they take a global point of view before selecting the packet with earliest deadline or minimum laxity.) The parameters that affect the schedulability in closed system are switch dimensions, schedule length, and average utilization of all links. In [15], Philp shows that the success rate drops rapidly as these three parameters increase[15].

Next, we use the same set of message streams, split them into two applications, each with link utilization up to 45%. (Two applications each asks for 50% of the total network capacity and each utilizes 90% of its requested bandwidth.) We apply EDF-RR, MLF-RR, EDF-SDR, MLF-SDR to each set independently and then combine the two schedules together as described in the previous section. We discover that the success rate improve dramatically using the hierarchical scheduling scheme. Even though we apply the same heuristic algorithms in the high level application sched-

uler as in the closed system, but scheduling each set separately reduces the size of the scheduling problem and hence improves the success rate.

4.3.2 Number of Applications and Load Distribution

We examine how the number of applications in the open system affects the schedulability of the network. For this experiment, we simulated systems with two, three, four, and five real-time applications. Moreover, we allow the load distribution of each system to vary. Table 4.3.2 lists the required capacities of the applications in each system. All message streams are periodic with precise release times. Again, the link utilization is 90% and the switch size is 4x4. Figure 4.6 shows the success rate in each system. The system that contains one application is a closed system. For system with two or more applications, the black color bar is the success rate when the network capacity is equally divided among the applications; the light gray bar shows the success rate when the required capacities of the applications are harmonic fractions of the total network capacity. (We intend to simulate a spectrum of capacity requirements and harmonic value is chosen for the sake of simplicity.) The shaded bar shows the success rate when one application dominates in the cluster while the rest of the applications only utilize a small fraction of the network bandwidth. We see that the success rate increases as more applications are allowed to run in the system. This is to be expected since each *HLS* has a smaller number of messages and the heuristic algorithms perform better. Furthermore, we observe that the load distribution makes a difference in the success rate as well. Among systems with the same number of applications, the success rate is highest when the network capacity is equally distributed among applications. For example, in a system running four real-time applications, the success rate of scheduling messages from applications each sharing 1/4 of the total bandwidth is substantially higher than that of one application is requesting 3/4 of the bandwidth while the rest each requests for 1/20, especially for the simpler Row-by-Row based algorithms. This is a mixed effect of schedule length and period variation.

In general, we observe that the hierarchical scheduling scheme allows a larger set of periodic messages to be scheduled in the network compared to a closed system. It reduces the scheduling

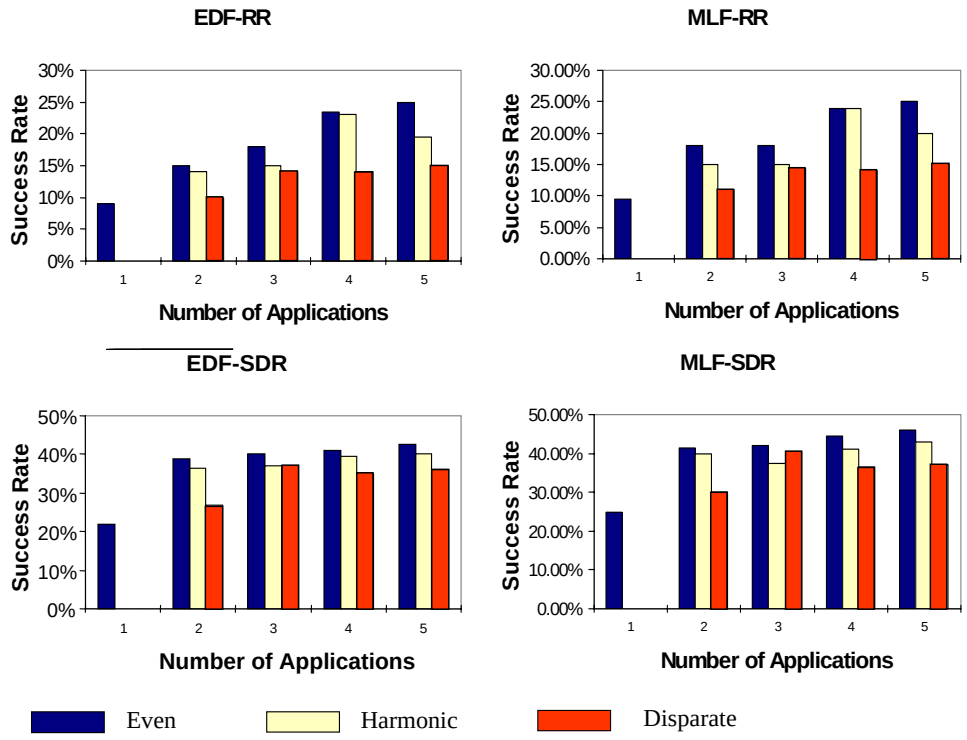


Figure 4.6: Success rates as the number of applications increases, N=4, U=90%, L=420

Table 4.1: System load distribution

Number of Applications	Required Capacity
1	Closed System
2	(1/2, 1/2), (3/4, 1/4), (9/10, 1/10)
3	(1/3, 1/3, 1/3), (1/2, 1/3, 1/6), (3/4, 1/8, 1/8)
4	(1/4, 1/4, 1/4, 1/4), (1/2, 1/4, 1/8, 1/8), (3/4, 1/20, 1/20, 1/20)
5	(1/5, 1/5, 1/5, 1/5, 1/5), (2/5, 1/5, 1/10, 1/15, 1/20)

problem size and yields a higher system utilization.

4.4 Implementation

We are currently implementing a prototype of the hierarchical scheduling scheme in a distributed environment which runs on a Myrinet cluster of PCs running Windows NT. A real-time application specifies its required network capacity, as well as the buffer size for its send and receive queues. If the application passes the acceptance test, a communication context is established and a conflict free schedule is generated. In essence, communication context is a partition of the LANai memory that is mapped directly into the application's address space. Messages are inserted into the send queue in the communication context according to a high level scheduling algorithm chosen by the application developer. The conflict-free schedule specifies which application is allowed to transmit in each time slot within a round. The Myrinet Control Program(MCP) running on each node loads the new schedule and resets the LANai card. After the communication context is setup, the application is switched to run in real-time mode. Packets in the ready queue of each HLS_k are transferred in the order given by the application's schedule under DMA control out to the network when the low level scheduler schedules application A_k to send.

4.4.1 Interaction between CPU and network scheduling

As discussed earlier, in order to provide end-to-end guarantee to applications, the interaction between CPU and network scheduling is crucial. The hierarchical network scheduling scheme guarantees that packet arrives at receiving host before its deadline. The prototype uses a user- level server, i.e. a communication server, to provide network services. To keep accurate accounting of the CPU resource, the communication server is executed by a passive server. A passive server is given only a very small amount of processor bandwidth and uses its own execution budget solely for administrative purposes. The processor time the communication server uses to perform any service is charged to the application requesting the service.

A real-time client application requests a service by calling one of two real-time application programming interface (RTAPI) calls, `SendData` and `RecvData`. When an application A_k calls the function `SendData`, it passes in the request the data to be sent and the handle to the server, as well as its own execution budget and deadline for consuming the budget. In response to the request, the communication server creates a new work thread in its own address space. The work thread inherits this budget and deadline. This thread is then inserted into the ready queue of the passive server on EDF basis. When the work thread is scheduled to run, it copies data into the communication context that has been reserved for the application. Also, it schedules the packets in the application's message queue according to the algorithm used by the application. Based on the required capacity the application specified and the length of each schedule cycle, each communication context receives credits to send at the beginning to each cycle. The *LLS* is implemented in the Myrinet Control Program. It sends $U_k'C$ packets every C slots from the message queue in application A_k's communication context.

A client application calls `RecvData` to request the communication server to process incoming data on its behalf. Similar to `SendData`, the `RecvData` function also causes a transfer of budget from the requesting application to the communication server. If the data to be received is not available when the requesting application calls `RecvData`, the call effectively freezes the requesting

application: the server of the requesting application has no budget, and the communication server holding the application's budget cannot execute on behalf of the application as the data is not available. To circumvent this problem, the application first makes a call to`WaitData` to indicate it is expecting incoming data. If the data is available, `WaitData` returns immediately. Otherwise, the CPU thread calling WaitData is blocked. Since `WaitData` function does not transfer budget to the communication server, other threads in the requesting application may still execute during the time when the calling job is blocked. When incoming data arrives, the communication server processes the message header and then notifies the receiving application about the source and length of the message. The application is expected to respond immediately. As soon as the thread calling `RecvData` is scheduled by the CPU scheduler and the application server is replenished, execution budget is transferred from the application to the communication server to process the packets and copy the message from the communication context to the application's address space.

The communication server is an important component in providing end-to-end guarantees in our design. It is responsible for conducting the network acceptance test, implementing the hierarchical network scheduler, and controlling the interaction between CPU and the network. The two-level network scheduler guarantees that the message arrives at its receiving host before its deadline, and the CPU scheduler guarantees that the CPU thread that is to process the message is dispatched before its deadline. A misbehaving application does not effect the schedulability of other real-time applications in the system, because each application has its own dedicated high level CPU and network schedulers and a separate communication context. The CPU budget replenishment scheme and the low level Myrinet scheduler together guarantees the end-to-end timing properties of the distributed applications in the system. Also, each application has its own message buffers in the communication context. An overload in an application and hence an overflow in one buffer would cause blocking and packet drops in the application alone and would have no effect on other real-time applications.

4.4.2 Implementation Issues

In our current implementation, there are several issues that require further study. Firstly, processors experience some clock drift. A skew between the different processors' execution of their schedules may cause network blocking. We make use of the feedback-based synchronization scheme proposed by Connelly and Chien [3]. However, the self-synchronizing schedule used in the feedback-based synchronization takes up several slots periodically to execute, e.g., in a 8-node cluster, the synchronization scheme takes 8 slots every 200 slots. We need to account this nonpreemptable section in our schedulability analysis. Secondly, the current scheme gives little support for non-real-time application. Unlike the total bandwidth server used for processor scheduling [6, 7], once a schedule is constructed, the slots reserved for real-time applications are left empty if the application has no packet to send. These slots cannot be used effectively for best effort traffic because the non-real-time traffic from one processor may block the real-time traffic from other processors in the cluster. Thirdly, for multi-switch networks, the complexity to construct global conflict-free schedules grows exponentially with the number of switches.

Chapter 5

Scheduling on Controller Area Networks

The requirement for reconfiguration operation is becoming increasingly important in modern industrial systems. This requirement has to be supported in the process control level as well as machine control level in the manufacturing industries, where fieldbus-based communication are commonly deployed. The Controller Area Network (CAN), in particular, plays an important role in applications where are timing requirements are stringent. In CAN-based distributed computer control systems, reconfigurability implies online addition, removal, and adaptation of message streams arised from control or monitoring application. Traditionally, reconfigurability and timeliness have typically been considered separately. CAN favors timeliness at the cost of expensive detailed global schedulability analysis and priority reassignment whenever reconfiguration is necessary. Hierarchical scheduling provides a new communication paradigm that achieve flexibility, timeliness, and efficiency in networked control applications.

In this chapter, we first give a brief overview of the CAN protocol. We then discuss the rationale in applying the open system framework on CAN-connected distributed system. We will describe a scheduling scheme that can be used to arbitrate and preserve bandwidth for each applications with both periodic and sporadic messages. A simple and effective admission control test is provided. Finally, simulation study is presented to validate the hierarchical scheme and show the scheduling achieves efficient bandwidth utilization while avoiding the complexity of the deadline encoding problem.

5.1 CAN Overview

The Controller Area Network is a high-integrity serial data communications fieldbus that operates at speed from 20Kbps to 1Mbps. The maximum bit rate of the CAN bus is limited by propagation speed of the signal and the bus length ($Bitrate_{max} = V_{signal}/2L_{bus}$).

CAN data is transmitted and received using Message Frames which is a non-preemptable transmission unit. The content of each message is labeled by an unique identifier. All the nodes on the network read the identifier and each performs a test to determine if the corresponding message is relevant. CAN protocol supports messages with 11 bit (Standard version 2.0A) or 29 bit (Extended version 2.0B) identifiers. Due to the overhead in the Extended version, the standard 11 bit format is by far the most widely adopted in existing applications today.

CAN uses the established method known as CSMA/CD but with the enhanced capability of non-destructive bit-wise arbitration to provide collision resolution and establish transmission order. During the start of an arbitration phase, all nodes with message to transmit send the identifier bits one bit at a time starting from the most significant bit. A logical AND is performed. If a node had transmitted a 0 but reads a 1, it knows there is another node with higher priority message to transmit and withdraw from the contention. The node that reads all its own priority bits from the channel wins the permission to transmit. This requires each contending nodes have a `unique` identifier. Figure 5.1 shows the format of a standard CAN message frame. Obviously, the higher the priority of the identifier is, the better QoS the message receives.

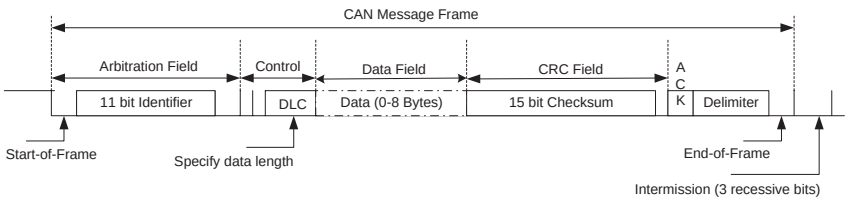


Figure 5.1: Format of the CAN message frame

5.2 Workload Assumptions

Many of the messages in a control application have periodic inter-arrival rates. Some other event driven messages have sporadic inter-arrival rates. But it is often possible to bound the minimum inter-arrival rates of sporadic messages. However, there also might be transient surges in network loading due to inaccuracies in the embedded devices or sudden detection of some safety critical events.

Therefore, we characterize the workload in a CAN system into four categories:

- hard real-time messages;
- hard real-time periodicmessages;
- soft real-time aperiodic messages; and
- non-real-time messages.

A periodic message M_i is characterized by a 3-tuple (m_i, T_i, D_i), where m_i is the message length[1], T_i is the period of the message[2], and D_i is the relative deadline of M_i. The ratio $U_i = m_i/T_i$ is denoted as the message utilization. We assume that each periodic message has a relative deadline D_i equal to its period T_i for the sake of simplicity. Sporadic and aperiodic messages are event driven. Their interarrival time and message length are identically distributed random variables. Sporadic messages have hard deadline D_i and the minimum interarrival time, in this case T_i, between any two messages are known. Soft real-time aperiodic messages are also expressed by the same 3-tuple except deadline misses can be tolerated.

[1] Assuming data payload is fixed, the message length is expressed as a number of frames required for sending the message.

[2] Each time unit is the time it takes to transmit one message frame which is a function of the frame size and network bit rate. T_i is expressed as a number of message frames.

5.3 Related Work

The conventional communication mechanism on CAN is centralized (e.g. *CANopen standard* [24]) which uses a master to poll its peripheral sensory devices periodically, makes decisions, and distributes commands to the actuators. The centralized control does not provide support for sporadic events and fails to exploit the distributed processing power in today's control networks.

In real-time scheduling research community, there exist several alternative approaches to the scheduling problem on the CAN bus. The most notable works are: 1) static priority mechanism proposed by Tindell and Burns [25]; 2) dynamic priority mechanism proposed by Zuberi and Shih [26].

Static priority protocols are simple to implement and bandwidth efficient when the traffic is periodic. Rate Monotonic (RM) and Deadline Monotonic algorithms are typically used to schedule the messages set. Under the rate monotonic algorithm, each message with distinct period should be assigned a distinct priority. In the context of CAN, the length of the identifier field limits the scheduling resolution as well as the number of messages supported by the system. The loss of schedulability can be reduced by employing a technique known as constant ratio priority grid [27]. For the rate monotonic algorithm, the percentage loss in worst case schedulability is provided in [28] as $Loss = 1 - (\log(\frac{2}{r}) + 1 - \frac{1}{r})/\log(2)$, r as the number of priority bits. However, not all 11 bits can be used for priority encoding since the identifier field also need to specify the destination address. This further reduce the scheduling resolution.

Dynamic priority schemes, on the other hand, is more efficient in the presence of sporadic messages. Shih et. al. proposed a scheduling method which encodes deadlines into a limited number of priority levels that tradeoff processor time to gain network utilization. However, dynamic priority approaches usually require the extended CAN protocol (with a 29-bit ID field instead of the 11-bit standard) and encounter high runtime overhead. The message ID can no longer be determined *a priori* and must contain the message destination and deadline. The 18 additional bits in the ID field account for 20-30% bandwidth overhead in a message with 1-8 bytes payload, which negates the

bandwidth efficiency obtained by the dynamic priority scheduling. Di Natale [29] also proposed a deadline encoding scheme which trades off computation complexity with network utilization. Moreover, as time progresses, deadline values may overflow the ID field and require a mechanism to update message ID periodically based on an increasing time reference. The complexity involved makes it difficult for the dynamic priority scheme to be adopted by the industry.

While each of the above mentioned scheduling approaches may be suited for a certain type of message set, we argue that they are over constraining in a sense that all timing characteristics (deadline, period, message length) must be known *a priori*. This is becoming increasingly difficult to comply. As the control applications become more and more complex, application software is more independent (or modularized). It is unrealistic to assume developers have global knowledge of all other applications that will be run concurrently in the system. Moreover, modern distributed real-time systems usually require a certain level of flexibility to accommodate on-line reconfiguration. These changes may include adding or removing applications, or changing the timing parameters of existing message set. Global schedulability analysis involves extensive verification of every combination of all the applications might be executed concurrently and must be done off-line. These requirements motivated us to apply the OPEN distributed framework on CAN.

In an open environment, the hierarchical schedulers allow us to combine different classes of algorithms to provide a practical approach to the CAN schedule problem. We will describe the scheduling scheme in details in the next section.

5.4 Hierarchical Scheduling Scheme on CAN

Scheduling on the CAN bus boils down to encoding the identifier of messages with different timing characteristics to get the most schedulable utilization. The hierarchical scheme for scheduling real-time messages fits in naturally in a CAN system. Grouping the message streams and schedule them through one or more servers in the local node enable us to best exploit local knowledge and relieves the burden of assigning an unique identifier to each messages in the network. Having a low level

server that queues the highest priority message from each high level servers, and controls the rate of message injected into the CAN bus based on reservation, give us the advantage of timing isolation and simplified admission control.

In the hierarchical scheduling framework, one sporadic server is created as the low level scheduler *LLS* to schedule all real-time messages generated on each node. There have been many different sporadic s erver algorithm folliwng the initial work by Sprunt et al. These various algorithms attempt to minimize the response time of soft-real-time messages They are distinguished by how aggressive they are at schedle soft-real-time messages. Some of the best known work include simple sporadic server [8, 30], deferrable servers [31], slack-stealers [32, 33], and sporadic servers [34, 35]. None of the servers performed consistently better than the others. The selection of sporadic server depends on the workload. For simplicity, we will assume the use of simple sporadic server for the time being.

The number of nodes in CAN tend to be limited, much less than the number of messages. Hence, it's not unrealistic to assume that each node has knowledge of the sporadic server reservations for all the nodes in the system. The reservation information is updated and broadcast in CAN bus whenever a sever in one of the nodes passes the admission test with new or updated reservation request. Each local scheduler can then construct a contention free schedule locally using an agree upon algorithm. The sporadic server with period p_s and budget c_s is treated exactly like a periodic message with a minimum interarrival time p_s and maximum execution time c_s. EDF is used to maximize utlization. Because contention is resolved implicitly, this algorithm is called Implicit-EDF (I-EDF). To illustrate an example, suppose each node is given a message table as shown in Figure 5.2, the same schedule is derived by every node in the cell according to EDF (deadlines ties are broken in favor of the node with the lowest node ID).

Non-real-time messages in the same node are served in FIFO order. However, a fair mechanism can be designed to establish order between the nodes. A priority level is assigned to the non-real-time queue in each node. The node ID can be used as the priority at time t_0. After each successful transmission of a non-real-time message, the priority at each queue is incremented by one. When

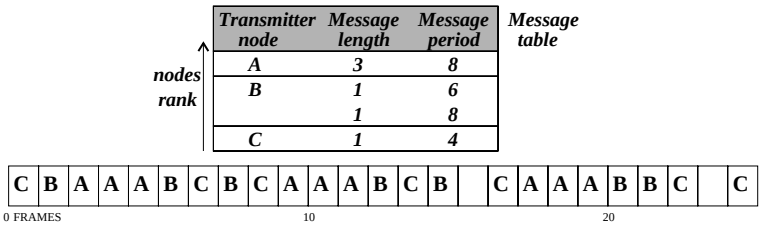


Figure 5.2: Example of implicit contention using EDF.

the priority becomes greater than the number of nodes on the CAN bus, it wraps around and starts over at priority level 1. Whenever a slot is not occupied by real-time and soft-real-time messages, the non-real-time message queue with the highest priority has the right to transmit. This is equivalent to rotating a token between the nodes. Each node has a fair access to the channel.

5.4.1 Encoding CAN Identifier

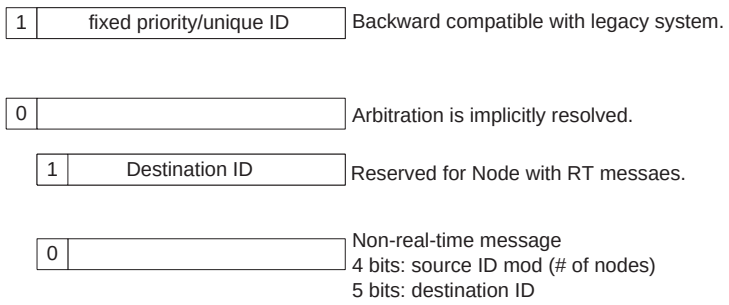


Figure 5.3: CAN identifier encoding: Scheme A

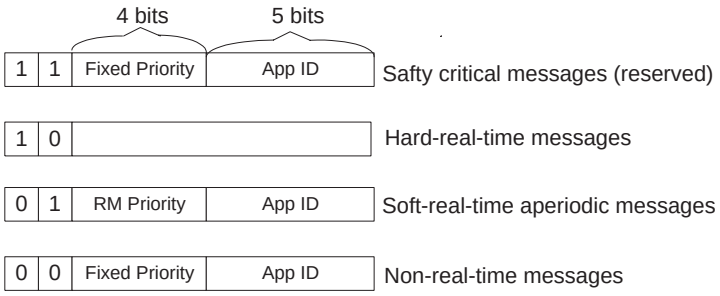


Figure 5.4: CAN identifier encoding: Scheme B

Figure 5.3 and Figure 5.4 show two schemes to encode the message identifier. With scheme A, setting the most significant bit to 1 indicates that the following 10 bits in the identifier contain

priority and destination ID of the message. The bits should be interpreted as in the legacy system. The nodes are then required to switch back to the fixed priority encoding system. 111...11 is reserved for emergency message which has the highest priority. Setting the most significant bit 0 means that I-EDF is being used. Each node on the network maintains its local EDF schedule. If it is scheduled to transmit, the node send its identifier bits 01, otherwise it stays quiet. The remaining 9 bits are used to encode destination address. Given the fact that the number of nodes connected to a CAN bus tends to be small, 9 identifier bits may seem like an over-allocation for destination encoding. But since bus contention can now be resolved implicitly without loading the identifier, we can afford to use these identifier bits for more advanced encoding schemes such as multi-casting. (While multi-casting support on CAN bus is outside the scope for this dissertation, it is definitely a future extension that is worth investigating.) If a node has non-real-time messages to transmit, it encodes the first two bits in its identifier 00. If it hears a 01 from the network, it knows that a real-time message is being transmitted and drops out from the contention. Otherwise it will continue to contend for channel access with the priority level of its non-real-time message queue. 4 bits are used to encode the priority level and remaining 5 bits are for destination address. Non-real-time messages cannot be multi-casted. Notice that scheme A does not distinguish between hard and soft real-time messages.

Scheme B, on the other hand, always reserves the two most significant bits to indicate the type of the message. 11 is for safety critical message, 10 is for hard-real-time, 01 is for soft-real-time aperiodic messages, and 00 is for non-real-time. The remaining 9 bits are used to encode priority level (4 bits) and destination ID (5 bits). In the same way as in Scheme A, hard-real-time messages exploit the remaining remaining 9 bits in the identifier for multi-casting. Any node that has soft-real-time or non-real-time messages to transmit, it sends a 01 or 00, respectively. If 01 wins the contention, the nodes with soft-real-time messages will sends their priority levels according to the rate monotonic algorithm. Otherwise, non-real-time messages will content for channel access in the same way as in Scheme A. Scheme B uses I-EDF to schedule hard-real-time message, RM for soft-real-time, and round-robin for non-real-time messages.

Both encoding schemes can be used in the open system and each has its own merit. Scheme A is more free-formed and provides better backward compatibility. Scheme B supports soft-real-time aperiodic messages and allows the aperiodic messages to be scheduled in a rate-based server along the with hard-real-time messages. In the next section, we will describe two algorithms that aim to improve the average aperiodic message response without compromising the timeliness of hard-real-time messages.

5.4.2 Improving Aperiodic Message Response Time

Two heuristic algorithms were proposed to improve the aperiodic message response time:Earliest Deadline as Late as Possible (EDL) and Implicit-EDF with slack stealing (I-EDF-SS). The basic idea is to delay the transmission of real-time messages as late as possible, so earlier slack can be used for aperiodic messages. Slack stealing in priority-driven system has always been an expensive operation. But the goal here is not to design an optimal slack computation algorithm. There is no way to suspend or active the slack stealer because the scheduler does not have global knowledge of when an aperiodic message is ready for transmission. Hence the heuristic is to get as much slack as possible at all times. Then bitwise arbitration is used to allow different nodes compete for the use of the slack to transmit the aperiodic message with the highest priority.

Figure 5.5 shows an example of a schedule constructed with the earliest deadline as late as possible heuristic. Essentialy, the heuristic is to schedule the periodic reservation with the earliest deadline in the $D-C$ slot. In the example, the first instance of (C, 1, 4) is scheduled first in slot 3, (A, 1, 6) in slot 5, and (B, 1, 8) in slot 7. Hence slot 0-2 are open to aperiodic traffics.

Figure 5.6 illustrate a scheme to schedule the message based on the reservation table in Figure 5.5. The hard-real-time messages from node A, B, and C are all scheduled before their deadlines. Two periodic “reservation”, (D, 1, 3) and (E, 1, 8), are added and scheduled using the Implicit-EDF algorithm. We refer to them as the slack stealing tasks. In the local schedule each node maintains, whenever the node see a slot for D or E, the *LLS* select a message from soft-real-time or non-real-time message queue to transmit. All the soft or non-real-time messages selected

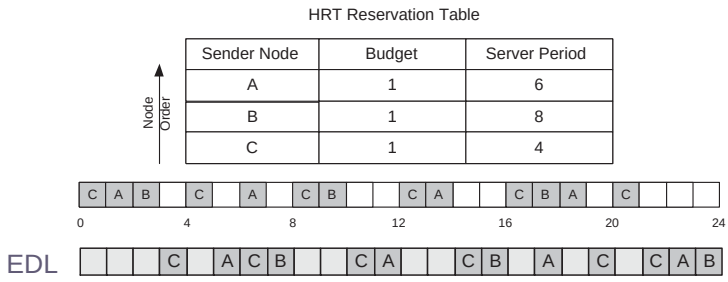


Figure 5.5: Schedule constructed with the Earliest Deadline as Late as Possible heuristic

to transmit will content for the bus access using their identifiers as encoded using the scheme described in the previous session. The remaining problem is to figure out how to derive the slack stealing reservations.

Since all the real-time messages are scheduled using the EDF algorithm, we can calculate the exact unreserved utilization. We can write this unreserved utilization in the form of the sum of two fractions, i.e. $C_{s1}/P_{s1} + C_{s2}/P_{s2}$. With some math manipulation, it is easy to find a combination such that P_{s1} is as small as possible. Using the example in Figure 5.6, the unreserved utilization is 11/24, $P_{s1} = \lceil \frac{24}{11} \rceil = 3$. C_{s2}/P_{s2} is 11/24 - 1/3 = 1/8. We then have C_{s2} is 1 and P_{s2} is 8.

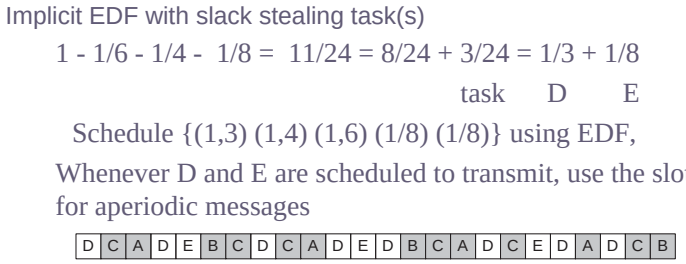


Figure 5.6: Schedule constructed with EDF with slack stealing tasks

These two heuristics are not optimal and give different slack times. However, they are very simple to implement and is effective in reducing the average response time for aperiodic messages without compromising the real-time messages.

5.5 Admission Test

Admission Test for CAN is very straightforward. Recall the reservation hierarchy depicted in Figure 3.3. At each level, round up the effective utilization if the period of the server is decreased, and then sum the total utilization at the MAC level. If the sum of all the $U_j' \leq 1$, accept A_k.

5.6 Simulation Study

In order to evaluate the performance of the sporadic server based approach, we performed a number of simulations. All experiments assume there are 8 nodes connected to the CAN hence 8 servers, a network speed of 125bits/ms, and data size is 8 bytes data, all are standard parameters for CAN network. In the first experiement, the server-based approach is compared against ideal EDF and RM under various workload. For the sake of simplicity, workload consist of all periodic hard-real-time messages.

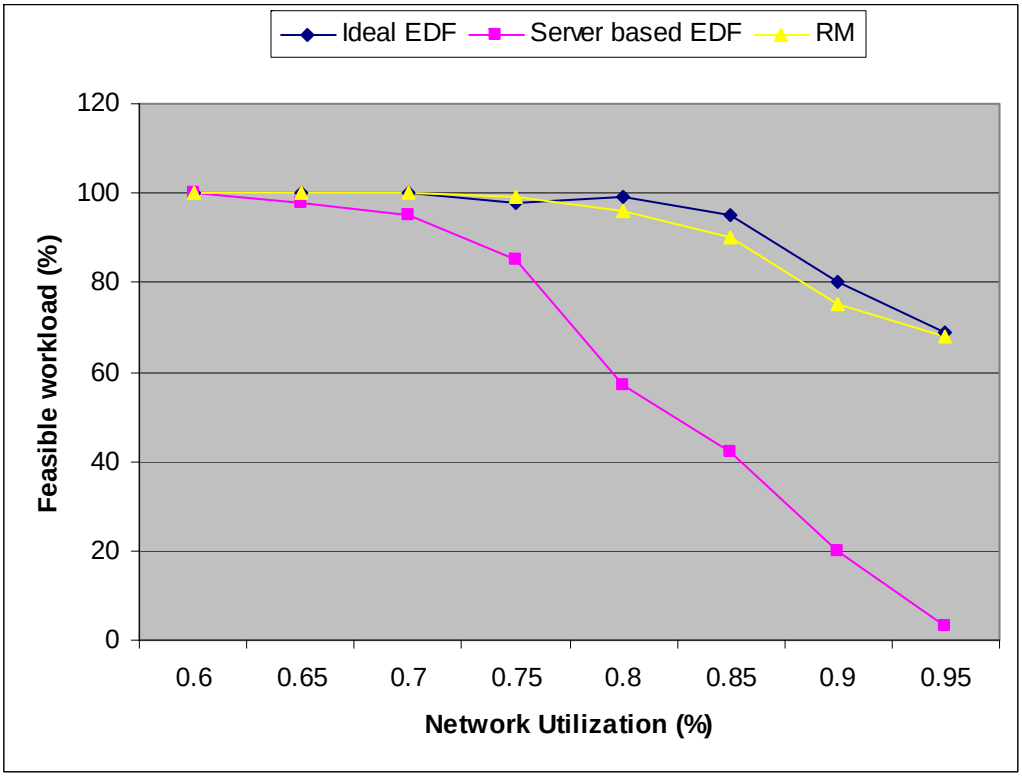


Figure 5.7: Percentage of schedulable messages with period between (10ms, 50ms)

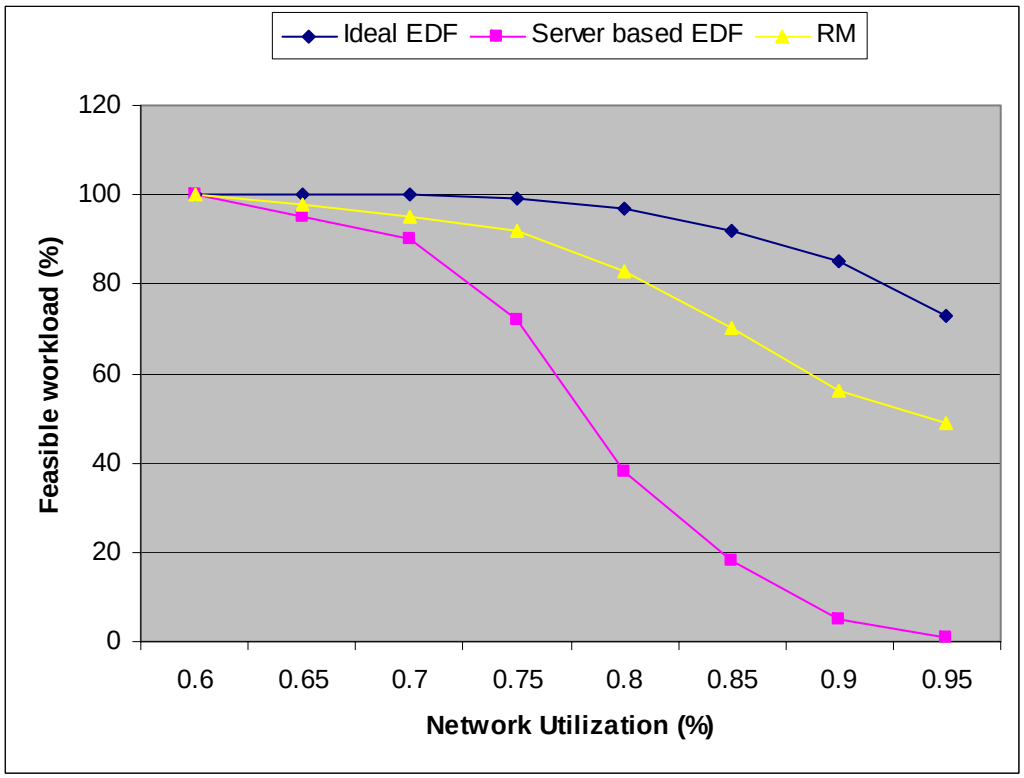


Figure 5.8: Percentage of schedulable messages with period between (10ms, 100ms)

Chapter 6

Energy Efficient Real-Time Scheduling on IEEE 802.11 Wireless LANs

This chapter is concerned with the problem of scheduling real-time messages in IEEE 802.11 Wireless LANs. Similar to CAN, WLAN is a broadcast network with constraints on network bandwidth. However, wireless network does not have the bitwise arbitration property. Instead, it relies on either a differentiated backoff scheme (based on weight) or a coordinator for medium access control. Each access mechanisms is better suited for certain type of traffic while lacking in others. We will study the MAC protocols more closely and attempt to achieve a relatively optimal algorithm that can support a diverse set of traffics.

In addition to the delay and bandwidth constraints of wireless channel, another severely constrained wireless resource is the limited battery life of portable wireless devices. The energy consumed by the transceivers of these portable devices is an important limiting factor in the amount of functionality that can be placed in these devices. There has been substantial advances in the hardware aspect of mobile communication energy efficiency, such as low-power states, and energy efficient modulation. However, further energy efficiency gain can be achieved in the higher layer of the protocol stack. One possibility is to trade CPU/memory/storage resources for better bandwidth allocation and energy consumption. An adequate objective in such an environment is then to meet the required temporal constraints of diverse applications, while minimizing the amount of energy used by the wireless transceivers of portable devices.

Specifically, we develop a centralized energy efficient algorithm that is in compliance with the hybrid coordination function (HCF) defined in IEEE 802.11 to support temporal QoS, called *scheduled contention free burst* (S-CFB), that

1. provides deterministic timing guarantee to real-time messages in the presence of non-real-time traffics;

2. allows independently developed and validated applications/devices to share the wireless medium without conflict;

3. minimizes the control overhead (of HCF) by bundling periodic transmissions into bursts; and

4. minimizes power consumption by generating schedules that facilitate devices to turn off their transceivers when they are not scheduled to transmit.

6.1 Brief Overview of IEEE 802.11 Standard

IEEE 802.11 has two basic access schemes: the Distributed Coordinator Function (DCF) and the Point coordinator Function (PCF). DCF is based on carrier sense multiple access with collision avoidance (CSMA/CA) and uses an optional RTS-CTS handshaking mechanism to reduce packet collision. PCF, on the other hand, is a centralized polling-based access mechanism that aims to support real-time traffic. In the current PCF standard, the Point Coordinator (PC) alternatively generates Collision Free Periods (CFP) and Collision Periods (CP).

Recently a separate subgroup within the IEEE 802.11 working group, is formed to focus on the MAC layer and expand capabilities and efficiency of the current access mechanisms for applications with QoS requirements. The task group has produced two draft schemes, EDCF and HCF, to replace the current DCF and PCF respectively. EDCF defines multiple priority classes and sets different minimum congestion window sizes and interframe spaces for classes of different priorities. However, there are only a limited number of priorities (8) and contention may still occur among connections in the same priority class. HCF, on the other hand, is proposed to remove

the alternating CFP/CP structure in PCF, as the periodic alternation between CFPs and CPs is not flexible enough and makes it difficult to accommodate multiple connections of different temporal requirements. HCF allows the PC to generate *Contention Free Bursts* (CFBs) as needed. Real-time traffics can now gain transmit opportunities in both the CP and the CFP. With the added flexibility provided by HCF, we can now incorporate more dynamic scheduling algorithms in the 802.11 MAC layer. For example, there are a variation of EDF-based dynamic resource allocation schemes that emulate general processor sharing, e.g., weighted fair queuing [36] and its variations. However, these algorithms are expensive to implement, as the instantaneous state information, such as virtual start/finishing time, is required for every scheduling decision made by the coordinator. As a result, the coordinator has to poll for every scheduled message. On the other end of the scheduling spectrum, simple but less efficient, the round robin (RR) and weighted round robin (WRR) schemes fix round length and transmission order in each round. We propose a middle ground solution that takes into account of elastic features of multimedia real-time traffic and balances between complexity and efficiency.

Typically, the transceiver can be in one of the five states: off, sleep, idle, receive, and transmit. Table 6.1 lists the average amount of energy consumed in each state by a WaveLAN interface card with a throughput of 2 Mbps (according to the AT&T WaveLAN specs [37]). It is obvious that maximizing the sleep time of the transceiver will significantly increase the energy efficiency of portable devices. However, the transition time between the sleep state and the other ON states is non-trivial. For example, it takes 80μs for a WaveLAN interface card to enter the sleep state and 250μs to wake up. Therefore, a device should turn off its transceiver only when the time until its next transmission is sufficiently large. In order to reduce the number of transitions and to maximize system utilization, it is more efficient to aggregate data packets and transmit them without preemption.

We will show that S-CFB can provide deterministic guarantee to real-time connections and yet allows the wireless devices to switch to the power-saving sleep state for a longer period of time. Moreover, the control frames required to maintain contention free transmission is reduced.

Table 6.1: Average power dissipation in operation states

State	Transmit	Receive	Idle	Sleep
Power (mW) Dissipated	3000	1675	1425	80

6.2 Variable Periodic Task Model

We focus on the scenario where a base station is present to coordinate access between nodes in an wireless LAN. In particular, we consider a SOHO (small office home office) environment in which the coverage area is small, nodes are relatively static, and the channel condition is less variable.

Real-time message is usually characterized by (C_i, P_i), where C_i is the message length and P_i is the period, both expressed as a number of slots.Real-time message (as well as the behavior of a simple sporadic server) by (C_i, P_i). However, in many real life applications, the sampling period P may be selected from a range of values depending on the available bandwidth and the different encoding schemes used to encode packets. For example, H.263 [38] employs motion estimation techniques that encode motion files at 30/15/10/6 frames per second. Most video clips streamed on the Internet today are coded at the 160x120 resolution which is in the 20-56 kbps range. High resolution videos can request up to 200 kbps bandwidth. The actual transmission bit rate can be adapted according to the available network bandwidth, as well as the end user preference on video resolution and transmission power consumption. To characterize this elastic feature, we model each periodic reservation request as (C_i, $P_{i,min}$, $P_{i,max}$), where $P_{i,min}$ and $P_{i,max}$ are the minimum and maximum allowable sampling periods and P_i' has to be chosen within the range of $P_{i,min}$ and $P_{i,max}$.

Choosing a set of appropriate P_i' is similar to the problem of selecting a set of control task frequencies to optimize system performance which has been addressed by Seto, Lehoczky, Sha and Shin in [39], and in the *Elastic Task Model* proposed by Buttazzo, Lipari and Abeni [40]. The elastic model exploits the flexible rate of adaptive control applications to keep the system underloaded. This proposed algorithm for WLAN, on the other hand, focuses on maximizing energy efficiency.

6.3 Scheduled Contention Free Burst Algorithm (S-CFB)

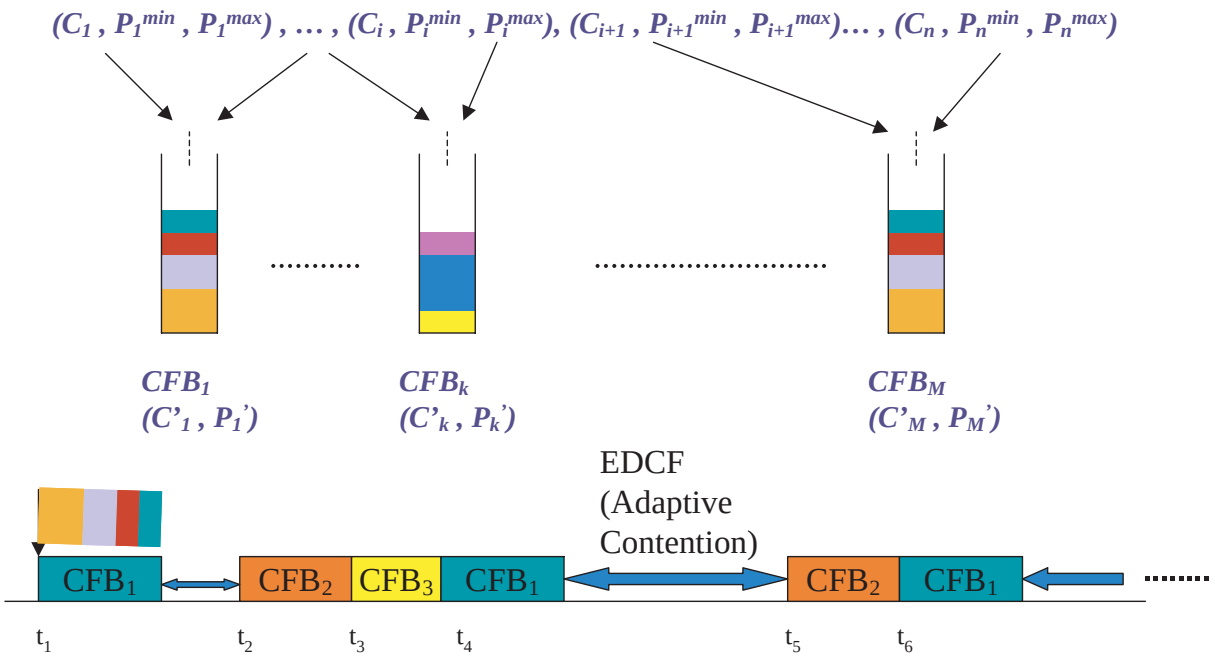


Figure 6.1: The operation of Scheduled Contention Free Burst

Figure 6.1 illustrates the basic idea of this scheme scheme. Each node reserves a fraction of the network bandwidth from a point coordinator (PC), which typically co-located with the base station. The PC divides the real-time messages into a number of groups [1] and transmits the real-time messages in each group in according to a predefined schedule. If a dynamic scheduling algorithm is used, the PC will either broadcast the schedule at the beginning of the group transmission period, or poll each node individually. In the interest of minimizing control overhead, we propose to use the WRR algorithm to produce a static schedule, which can be stored at each node and updated only when a new message stream is added or removed from the group. Both the static and dynamic schemes guarantee that the transmission is contention free. In the IEEE 802.11 HCF standard, these contention free periods are referred to as *CFBs*. The CFBs, in turn, are scheduled as periodic

[1] All messages in the same group have the same period P_i', which are selected in their overlapped period range. In the rest of the paper, the group period is denoted as P_k'.

messages[2] according to the EDF algorithm.

In order to maximize the amount of time a node can stay asleep and hence minimize the state transition (SLEEP to ACTIVE, ACTIVE to SLEEP) cost, the message transmission in each contention free burst is non-preemptable. The objective is to manage the CFB_k in such a way that, subject to the schedulability constraint, the number of CFB_k is minimized, and the optimal value of P'_k is chosen to as close to the $P_{i,max}$ value as possible. The rationale behind this objective will discussed further in the next section. Note that if P'_k is smaller than $P_{i,max}$ for certain request R_i in CFB_k, additional bandwidth will be allocated to that connection.

After CFB's are selected and assigned, the coordinator notifies each mobile device of the CFB it belongs to, the transmission schedule and the start of the next burst time of the CFB. Note that this information only need to be broadcast once when the coordinator accepts a new flow into the network. The transmission schedule is fixed once (C'_k, P'_k) is assigned for each CFB. Each node knows its position in the transmission sequence, wakes at a fixed number of slots after the start time of its scheduled burst, transmits the message without preemption, and then switches back to the sleep state. We refer to this scheme as the *Scheduled Contention Free Burst*, or *S-CFB*.

Finally, EDCF can be used in between the CFBS to transmit soft and non-real-time traffics. As mentioned earlier, EDCF supports differentiated service by defining multiple priority classes, which is ideal for messages that prefer some level of QoS but do not require deadline guarantee.

6.3.1 An Example

Given a set of periodic reservation requests {(10, 20, 40), (6, 45, 60), (5, 55, 65), (5, 80, 100), (10, 90, 120)}, Figure 6.2 gives the transmission schedule constructed using three different algorithms: earliest-deadline-first (EDF), weighted-round-robin (WRR), and S-CFB. EDF gives the most efficient schedule in terms of bandwidth utilization. However, there exists no regular pattern in the schedule, and hence each node in the network would need to stay awake all the time listening for the POLL message from the coordinator. WRR, on the other hand, constructs a fixed schedule in

[2] CFB_k is characterized by a group period P'_k and message length equal to $\sum_{M_i \in CFB_k} C_i$.

each round. Each node knows the exact time it needs to wake up to transmit and sleeps through the rest of time in each round. While WRR is much more energy efficient and simple to implement, it yields poor bandwidth utilization. In order to meet the timing constraint, the length of each round is limited to the $T_{i,max}$ value of the connection with the highest priority (40 in the example). All connections, including those that do not have messages ready in a round, are assigned one or more slots in each round based on their reservation requests. (For example, at time $t = 50$ and 56, R_2 and R_3 have no messages to transmit.) These over-reserved slots are a waste of bandwidth, especially when the values of T_is in system vary dramatically.

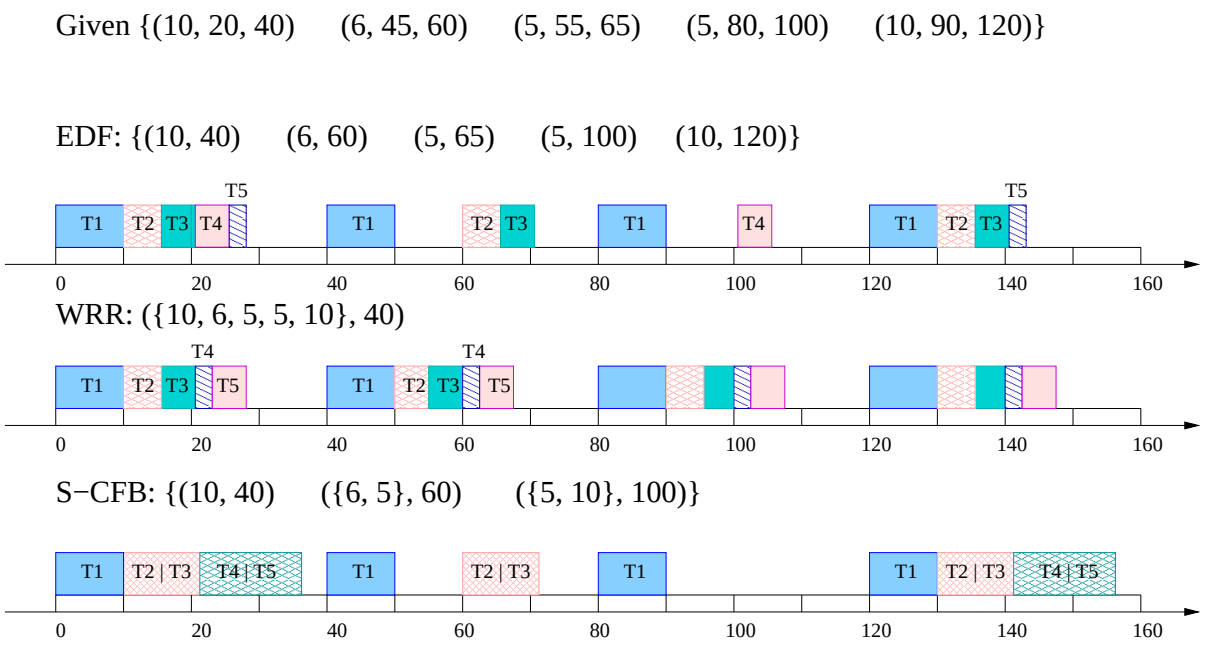


Figure 6.2: An example with different scheduling algorithms

S-CFB is a hybrid of EDF and WRR algorithms. Connections are bundled into contention free bursts as shown in Figure 6.2. The three CFBs are scheduled by the EDF algorithm. Packets in each CFB are scheduled using WRR. $t = 10$ is the start time of CFB_2, and R_2 and R_3 are assigned 5 and 6 slots, respectively. The next instance of CFB_2 is at $t = 60$ when both connections have packets to send. In this manner, no slots are wasted and nodes are allowed to sleep until their next scheduled contention free burst.

6.4 Formal Formulation

Now we are in a position to formally define our problem. Given a set of periodic message reservations **M** = $\{M_1,...,M_n\}$, where R_i is represented by a 3-tuple (C_i, $P_{i,min}$, $P_{i,max}$). Group reservations in **M** into a set of aggregated CFBs, **CFB** = $\{CFB_1, ..., CFB_m\}$, where CFB_k is represented by (C'_k, P'_k), $C'_k = \sum_{M_i \in CFB_k} C_i$, and $P_{i,min} \leq P'_k \leq P_{i,max}, \forall M_i \in CFB_k$. Without loss of generality, we assume $P'_i \leq P'_j$, if $i < j$. **CFB** is scheduled using EDF, and connections in each CFB are scheduled using WRR.

The objective is to determine **CFB** such that the system utility is maximized along two QoS dimensions: power consumption and bandwidth utilization. One conventional multi-objective optimization technique is the weighted sum approach. More specifically,

$$\bar{U} = w_p \times U_p + w_{bw} \times U_{bw}.$$

$\bar{U}$ is the overall system utility; U_p and U_{bw} are the user specified utility functions for power consumption and bandwidth utilization; w_p and w_{bw} are the respected weight. However, because all the parameters U_p, U_{bw}, and w's are not readily available, we simplify the objective by making the following observations:

- U_p increases monotonically as the power consumption decreases.
- Determination of U_{bw} is not so straightforward, as it depends on the user preference and network state. If the network load dynamically changes, it is desirable to minimize bandwidth allocation in order to accommodate new connections. However, some application may require a better user perceived quality and hence high allocation whenever bandwidth is available. In this case, these applications may specify a higher EDCF priority level and request to transmit in the contention period as well as in the CFB.
- As data transmission consumes power, it is desirable to reduce the transmission rate, i.e., P'_k should be chosen as close to $P_{i,max}$ as possible, $\forall M_i \in CFB_k$.

- As mentioned earlier, it is also desirable to maximize the sleep time of the transceiver and yet ensure the transceiver is turned off only when the time until the next transmission is sufficiently large. This implies more connections should be bundled into the same contention free burst and transmit without preemption and the number of CFB's should be reduced.

Based on the above observations, we should attempt to minimize both the power consumption and the bandwidth allocation. However, the two objectives may sometimes conflict with each other. Since we primarily focus on the energy consumption issue, we proceed in two steps: (i) we minimize the energy consumption by bundling reservation requests into a minimum number of CFBs; and (ii) we determine a set of bandwidth allocation that minimizes the bandwidth utilization.

6.4.1 Schedulability analysis

Next, we derive a sufficient schedulability condition for the real-time reservation requests. As stated earlier, the CFBs are scheduled as periodic messages according to the EDF algorithm. The classical Liu and Layland condition [41] guarantees that, in the absence of blocking, a set of periodic messages is schedulable with EDF if and only if

$$\sum_{k=1}^{n} \frac{C_k}{T_k} \leq 1,$$

However, in our model, the message transmission in each contention free burst is non-preemptable. Let B_k denote the longest message length of all CFBs other than CFB_k, that is $B_k = \max_{i \neq k}\{C_i'\}$. B_k is the maximum blocking time the CFB_k can suffer due to the nonpreemptibity of other CFBs. We make use of the schedulability condition for periodic tasks with non-preemtable sections [42]:

Lemma 1. *A CFB_k with a total blocking time B_k is schedulable with other CFBs according to the EDF algorithm if*

$$\sum_{j=1}^{n} \frac{C_j'}{T_j'} + \frac{B_k}{T_k} \leq 1, \forall k = 1, 2, ..., m. \tag{6.1}$$

[42]

Moreover, recall that the real-time messages in each CFB_k is transmitted according to the WRR algorithm. The period T'_k is within the period range of all the messages in the CFB_k. Therefore, the bandwidth utilization of CFB_k can be written as $\frac{\sum_{M_i \in CFB_k} C_i}{T'_k}$. It is straightforward to see the messages in CFB_k are schedule if CFB_k is schedulable. Following directly from Lemma 1,

the schedulability condition for all real-time messages in the network is as follow.

Lemma 2. *A set of independent periodic messages* $\mathbf{M} = \{M_1,...,M_n\}$*, if grouped into a set of non-preemtable CFBs,* $\mathbf{CFB} = \{CFB_1, \ldots, CFB_m\}$*, where* CFB_k *is represented by* (C'_k, T'_k)*, is schedulable according to the EDF algorithm if*

$$\sum_{j=1}^{m} \frac{\sum_{M_i \in CFB_j} C_i}{T'_j} + \frac{B_k}{T'_k} \leq 1, B_k = \max_{i \neq k}\{C'_i\}, \forall k = 1, 2, \ldots, m. \tag{6.2}$$

[42]

In summary, the CFB selection and assignment problem can now be stated as:

Step 1. Minimize the number, m, of CFBs.

Step 2. Minimize

$$\sum_{k=1}^{m} \frac{\sum_{M_i \in CFB_k} C_i}{T'_k},$$

subject to the constraint defined by Eq 6.2 in Lemma 2.

6.5 Optimal vs. Heuristic Solution

The QoS management optimization problems are known to be NP-hard[3]. All the optimal algorithms (of exponential complexity) essentially enumerate the solution space in some fashion. The optimal solution constructs an exhaustive search tree, with branches pruned as early as possible.

[3]The problem of managing one resource along one QoS dimension can be easily reduced to the 0-1 Knapsack problem which is NP-hard. The multiple dimension problem is at least as hard as the one dimension problem.

However, the worst case complexity of the algorithm is still $O(n!)$. As the exhaustive search algorithm is of exponential complexity, we propose in this section a $O(n^2)$ heuristic algorithm.

6.5.1 An Optimal Solution

ll the optimal algorithms (of exponential complexity) essentially enumerate the solution space in some fashion. To limit the search space, we first prove the following lemma.

Lemma 3. *In the optimal solution, the value of T_k' must be chosen from $\{T_{i,max} \mid M_i \in CFB_k\}$.*

Proof: *We prove by contradiction. Assume in the optimal solution $\tilde{M} = \{(C_i, T_k'), M_i \in CFB_k, 1 \leq k \leq m$, there exists at least one CFB, which we denote as*

$$CFB_j = (\sum_{M_i \in CFB_j} C_i, T_j'),$$

such that

$$T_j' \notin T_{i,max}, \forall M_i \in CFB_j.$$

Now we construct another solution $\widehat{M}$ by replacing T_j' with

$$\min_{M_i \in CFB_j} \{T_{i,max}\},$$

then $\widehat{M}$ is still a feasible solution (that satisfies the schedulability constraint) with the same value of m but a lower value of

$$\sum_{k=1}^{m} \frac{\sum_{M_i \in CFB_k} C_i}{T_k'}.$$

This implies that $\tilde{M}$ cannot be the optimal solution which contradicts the assumption. Hence, the value of T_k' must be in the set $\{T_{i,max} \mid M_i \in CFB_k\}$.

Our optimal solution constructs an exhaustive search tree, with branches pruned as early as possible with the use of schedulability constraints. The height of the tree is the number of connections in the system. Specifically, the exhaustive search algorithm consists of the following steps:

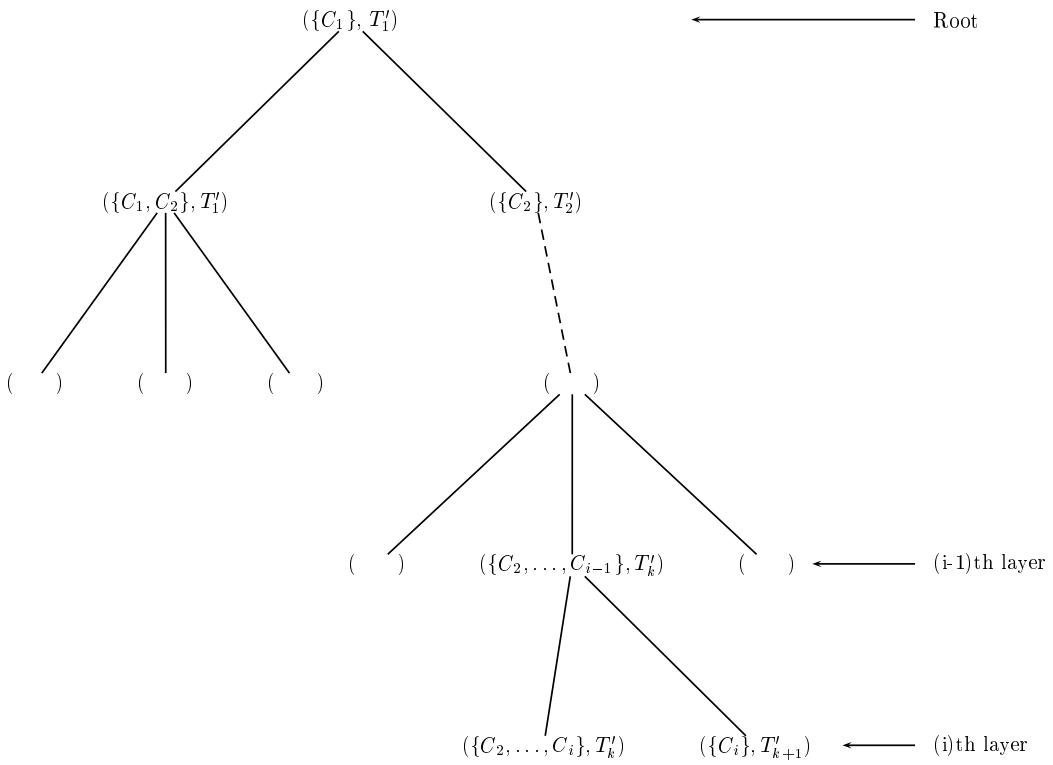


Figure 6.3: The search tree constructed under the optimal solution.

Step 1: Sort the connections in the increasing order of the $T_{i,max}$ values. Without loss of generality, we assume $T_{i,max} \leq T_{j,max}$ if $i < j$.

Step 2: Starting from the connection with the smallest value of T_{max}, (C_1, $T_{1,max}$), build a search tree to enumerate all possible solutions as follows:

Step 2.1: Assign (C_1, $T_{1,max}$) to CFB_1 and position it at the root of the search tree.

Step 2.2: Conduct the schedulability test to check if the utilization constraint (Eq. (6.2)) has been violated. If not, keep the root node; otherwise, remove the root node and terminate with no solution found.

Step 2.3: For each $j := 2$ to n, add the jth level child nodes to the tree by assigning (C_j, $T_{j,max}$) to be the leaf of some $(j-1)$th-level nodes. Specifically,

Step 2.3.1 For each existing node at $(j-1)$th level, (C_i, T'_k), $i \in CFB_k$: if $T_{j,min} \leq T'_k \leq T_{j,max}$, add a new jth level child node by assigning message M_j to CFB_k. Conduct the schedulability test. If the constraint is violated, remove the child node. Otherwise, keep the new child node.

Step 2.3.2 add a new jth level child node by creating a new leaf ($\{C_j\}$, $T_{j,max}$). Conduct the schedulability test. If the constraint is violated, remove the child node. Otherwise, keep the node and a new contention free burst, CFB_j, is created.

Step 2.4: Calculate the value of m for each leaf at the highest level of the tree.

Step 2.5: Among the nodes with the minimal value of m, find the node with the minimal value of

$$\sum_{k=1}^{m} \frac{\sum_{M_i \in CFB_k} C_i}{T'_k}.$$

The solution represented by the leaf node is the optimal solution.

Note that in the above enumerative algorithm, we attempt to assign a periodic reservation, $(C_\ell, T_{\ell,min}, T_{\ell,max})$, to one of the existing CFBs without violating the schedulability constraint. If

this is not feasible, we then insert $T_{\ell,max}$ into the list and reuse the tree from the $(\ell-1)$th level. However, the worst case complexity of the algorithm is still $O(n!)$.

Figure 6.4 shows how the search tree is generated in Example 6.3.1 using the exhaustive search algorithm. (C_1, $T_{1,max}$) is placed at the root. The left branch is pruned because $[T_{1,min}, T_{1,max}]$ does not overlap $[T_{2,min}, T_{2,max}]$. A new CFB is created with C_2' equal to 6, and T_2'. At the third level, M_3 can be either (i) bundled with M_2, creating a node with $T_2' = 60$ and $C_2' = 11$ $(6,5)$; or (ii) put into a new CFB_3 with $C_3' = 5$ and $T_3' = 65$. The process repeats until nodes at the 5th level are created. There exist four leaf nodes. For each leaf node, the value of m is calculated by tracing back to the root and finding the number of unique T_k' in the path. In this example, the first leaf node at the 4th level has the smallest value of m, 3, and represents the optimal solution of $\{$ ($\{10\}$, 40), ($\{6,5\}$, 60), ($\{5, 10\}$, 100) $\}$. It satisfies the schedulability constraint with U equal to $15/100 + 11/60 + 10/40 + 15/40 = 0.942 < 1$.

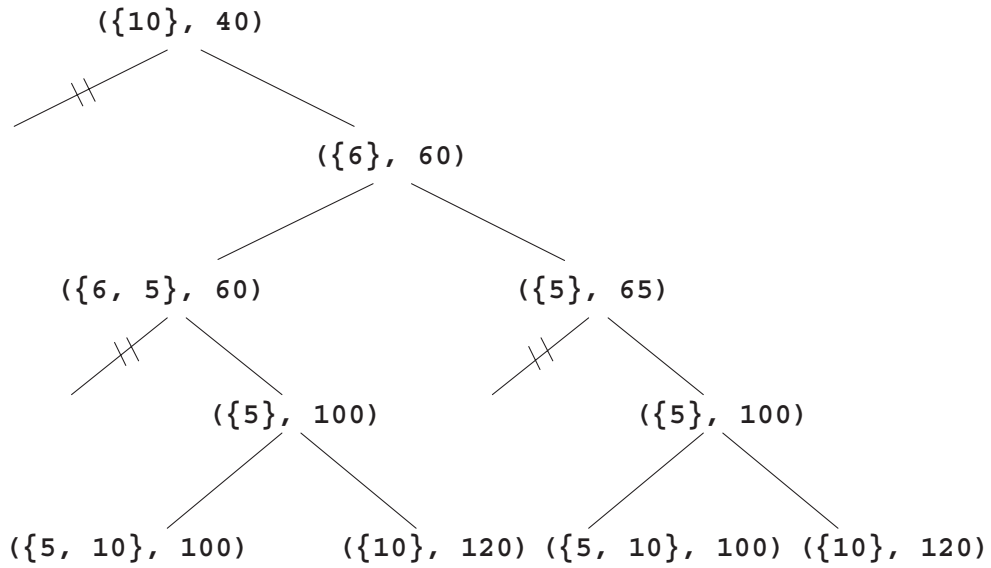


Figure 6.4: The search tree in Example 1 constructed using the exhaustive search algorithm.

6.5.2 A Heuristic to Approximate the Optimal Solution

The basic idea behind the heuristic algorithm is to divide the connections into a minimum number of CFBs, subject to the condition that connections in the same CFB must have overlapping P values. The P' value for each CFB is then chosen to be the largest $P_{i,max}$ value of all connections in the CFB. The schedulability test is conducted after the minimum number of CFBs are located. If the assignment violates the schedulability constraints, the algorithm then attempts to reduce the utilization by reducing the blocking term. That is done by slicing the CFB, CFB_k, with the largest value of C'_k into two (as evenly as possible without slicing C_i, $\forall M_i \in CFB_k$). The value of P' in each of the CFBs will be adjusted accordingly. The pseudo code for the heuristic algorithm is listed in Figure 6.5.

Step 1: Sort the connections in the increasing order of the $P_{i,max}$ values. Without loss of generality, we assume $P_{i,max} \leq P_{j,max}$ if $i < j$.

Step 2: Starting from the connection with the largest value of $P_{i,max}$ (i.e., $P_{n,max}$), set CFB_1 = (C_n, $P_{n,max}$).

Step 3: For $i = n$ downto 2 do

- **Step 3.1:** If $P_{i-1,max} > P_{i,min}$, add R_{i-1} to the current CFB and set $CFB_j \leftarrow$ (C_{i-1}+C'_j, $P_{i-1,max}$).
- **Step 3.2:** If $P_{i-1,max} \leq P_{i,min}$, start a new CFB (increment j) and set $CFB_j \leftarrow$ (C_{i-1}, $P_{i-1,max}$).

Step 4: Conduct the schedulability test on the set **CFB**. If the schedulability constraint is not violated, return **CFB**; otherwise, split the CFB with the largest value of C' evenly and repeat **Step 3**.

Figure 6.5: The pseudo-code of the proposed heuristic algorithm.

We use the same reservation set in Example 6.5.1 to illustrate the heuristic algorithm. As shown in Figure 6.6, we start at $S1 = 120$ and move toward the smaller values of Ts. At $S2 = 100$, $[T_{4,min}, T_{4,max}]$ starts to overlap with $[T_{5,min}, T_{5,max}]$. In order to put the two connections into the same CFB, the value of T'_1 must be set to $S2 = 100$. At $S3 = 90$ we reach one of the T_{min} values

of connections already in this CFB. Hence the value of T_1' is fixed, no other connections can be added to this CFB, and we complete the assignment for CFB_1 = ($\{C_5, C_4\}$, $S2$)=($\{5,10\}$, 100). The process repeats with $S4 - S7$ until CFB_2 = ($\{C_3, C_2\}$, $S5$) = ($\{5,6\}$, 60) and CFB_3 = (C_1, S_7) = (10, 40) are located. Then the schedulability test is conducted. In this example, $U < 1$ and { ($\{10\}$, 40), ($\{6,5\}$, 60), ($\{5, 10\}$, 100) } is the solution.

{(10, 20, 40) (6, 45, 60) (5, 55, 65) (5, 80, 100) (10, 90, 120)}

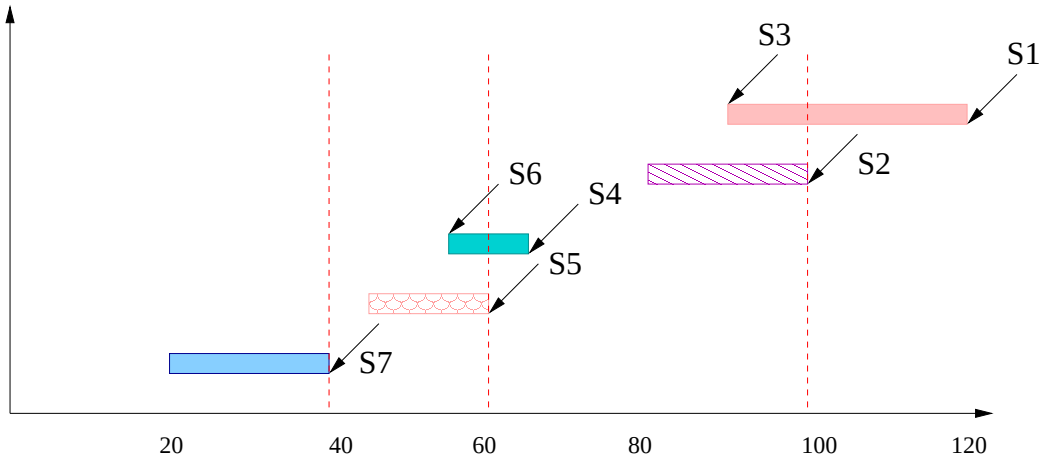


Figure 6.6: Example revisited to illustrate the heuristic algorithm

6.6 Open System on WLAN

Notice that that S-CFB algorithm proposed for the MAC scheduler is also a hierarchical scheme. The coordinator is now responsible for constructing the contention free schedule by grouping periodic reservation at each wireless node into CFB. CFBs are scheduled using EDF, and periodic reservations in a CFB are scheduled using WRR. Each node is guaranteed C_i slots of transmission in every P_k' ($M_i \in CFB_k$) slots. This extends the scheduling hierarchy by yet another level, but for a different purpose. Instead of providing isolation, this layer of abstraction trades of bandwidth utilization for energy efficiency.

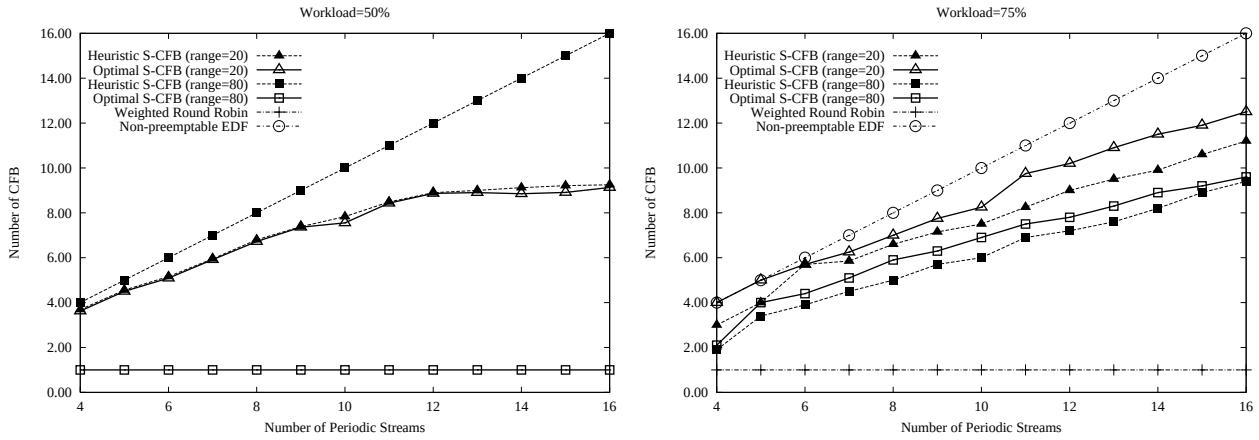
6.7 Simulation Study

In this section, we present the simulation results. We simulate scheduling of n periodic real-time streams at the coordinator of a wireless LAN. Each of the periodic connections (C_i, $P_{i,min}$, $P_{i,max}$) are randomly generated. Specifically, we first fix the overall system workload (U) to 50% and 75%. For each of the workloads, we generate n real-time streams. For a given workload (U) and a fixed number of periodic streams (n), we randomly generate $P_{i,min}$ and $T_{i,max}$ from a uniform distribution over the range of 200 to 1000. The value of C_i is chosen such that $C_i/P_{i,max}$ is equal to u_i which is randomly generated from a uniform distribution between [0, $2 \times U/n$], subject to $\sum_{i=1}^{n} u_i \simeq U$. We study how these parameters (U, n, C_i, $P_{i,min}$, and $P_{i,max}$) may affect the overall system schedulability as well as the power consumption.

The first set of experiment studies how well the proposed heuristic algorithm approximate the optimal algorithm. We also compare the S-CFB against other commonly used scheduling algorithm such as EDF and WRR. As mentioned in Section 6.2, S-CFB essentially strikes a balance between EDF and WRR, and hence we use them as the baseline schemes for comparison. The performance metrics used are: (i) the number of CFBs the coordinator generates given a set of periodic reservation requests, and (ii) the bandwidth utilization.

The second experiment is run in the ns-2 network simulator environment. We implemented the S-CFB by extending the IEEE 802.11 PCF module implemented by Lindgren et al. [43]. The PC maintains a table of CFBs and schedules the CFBs as periodic messages according to the nonpreemtive EDF algorithm. Each node keeps the schedule of the CFB it belongs to, wakes up before the CFB's expected transmission time, and sends its real-time messages according a static schedule generated by the WRR algorithm. The PC notifies each node of the CFB's next transmission time. We assume there is no non-real-time traffic for the time being. All the nodes go to sleep whenever they are not scheduled to transmit. The power dissipation values in different operation states listed in Table. 1 are used. We compare energy dissipation of our scheme against the 802.11 PCF with power saving mode. In order to provide real-time guarantee with the PCF, a

WRR-like algorithm is needed to determined the contention free period repetition interval length and the polling order. PCF is equivalent to the S-CFB scheme with 1 CFB. Results reported on energy saving are averages of 10 runs randomly-chosen scenarios selected from experiment 1.



(a) CFB generated under 50% workload. (b) CFB generated under 75% workload.

Figure 6.7: The number of CFBs generated by the optimal and heuristic S-CFB schemes.

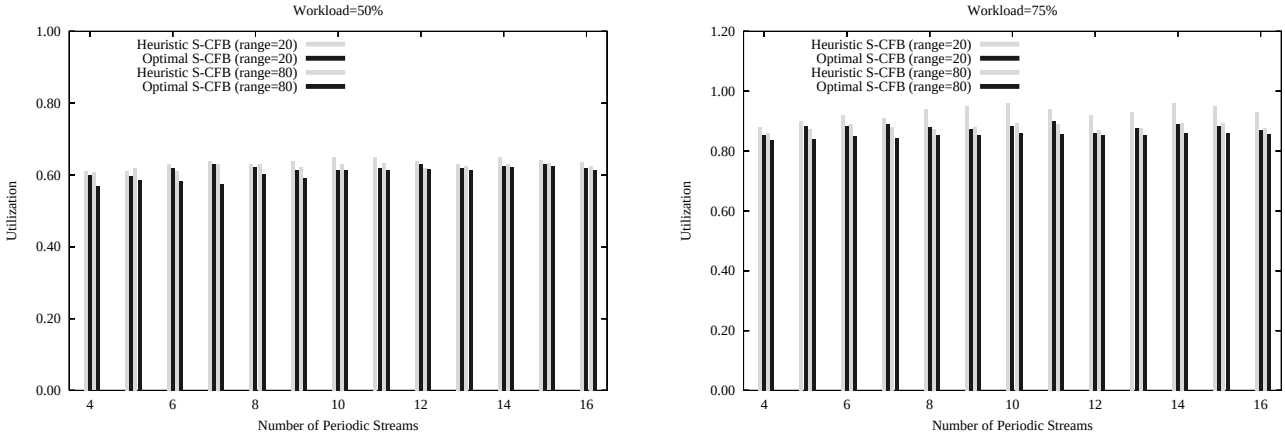
6.7.1 Results and Analysis

Figure 6.7 (a) and (b) gives, for a set of periodic real-time connections, the number of CFBs the coordinator generates under the light and heavy load scenarios, respectively. Several observations are in order: first, the straight line that corresponds to WRR is trivial because WRR requires that the one round length be smaller than the smallest value of T_is, for all the connection R_is in the network. EDF is assumed to be non-preemptable, so the number of transmission opportunities in each period is exactly 1 (assuming connections are backlogged). As expected, S-CFB falls between EDF and WRR. Second, the four lines represent the optimal and heuristic solutions found for a connection set with $\{T_{i,max}, T_{i,min}\}$ varying according to a uniform distribution of [0, 2 $\times$ range], where "range" is defined as $(T_{i,max} - T_{i,min})/2$. As shown in Figure 6.7, the heuristic algorithm approximates the optimal algorithm very closely. The connections with a larger "range" value (i.e., whose $[T_{i,min}, T_{i,max}]$'s tends to overlap with one another) are more likely to be bundled together and

hence give rise to a smaller number of CFBs. Third, when the workload increases to 75%, there exists some discrepancy between the line representing the optimal solution and that representing the heuristic solution. This is because when the network is relatively loaded, the number of tasks is small, and the overlapping "range" is small, join of a new connection would create a large blocking term which forces the CFB to be split into two. The resulting schedule resembles that generated by a nonpreemptive EDF. On the hand, if the range is large, it makes more sense to schedule them using WRR.

Figure 6.8 (a) and (b) shows the effect that certain amount of bandwidth is wasted due to bundling connections together and choosing a value of T' that may be smaller than some of the $T_{i,max}$ values. In particular, Figure 6.8 (a) shows an overhead around 20% (close to 65% of the bandwidth is consumed for bandwidth requests of 50%) under a light workload scenario. However, note that the overhead does not change significantly as the number of tasks varies. Also, the optimal solution gives better results than the heuristic solution. This is because the heuristic algorithm stops when it finds the smallest number of CFBs and one feasible schedule. The optimal algorithm, on the other hand, searches through the solution space for the minimum value of m and then the minimum value of U. The discrepancy between the optimal solution and the heuristic solution becomes even more pronounced in Figure 6.8 (b). This is because as the size of the connection set increases, the choice of "bundling" with other connections to form a new CFB is more. The packing scheme do not change the number of CFBs, but instead changes the blocking term (which is defined as the largest value of C' of all the CFBs with larger relative deadlines) and hence influences the value of U as well.

Finally, Figure 10 shows the energy saving factor of the S-CFB scheme over PCF. Energy saving is defined as the ratio of average power consumed by each node after 300 seconds of simulation using the PCF scheme and S-CFB scheme. It is reasonable to see the energy saving factor increase as the number of node increases. PCF performs reasonably well compare to S-CFB under light workload. S-CFB shows a much better energy efficiency as the number of nodes increases and as the workload increases.



(a) BW allocated under 50% workload. (b) BW allocated under 75% workload.

Figure 6.8: The amount of bandwidth allocated when the optimal and heuristic CFB schemes are used to schedule periodic workload.

In summary, we have presented, with the objective of efficient energy consumption, a scheduling framework for IEEE 802.11 wireless LANs. By exploiting the elastic feature of QoS requirements and bundling connections into contention free bursts (CFBs), this scheme reduces the amount of time portable devices need to stay awake, the control overhead in coordinating transmission of periodic real-time connections, as well as the number of state transition between the SLEEP state and the other ON states. We have conducted a simulation study in ns-2 which justified the energy efficiency claim. It is also in line with the concept of allowing independently developed applications to run concurrently without having to perform a detailed global schedulability analysis in the Open System Environment. We also validated that the proposed heuristic algorithm closely approximates the optimal algorithm through simulation. The heuristic is simple to implement and of polynomial time complexity.

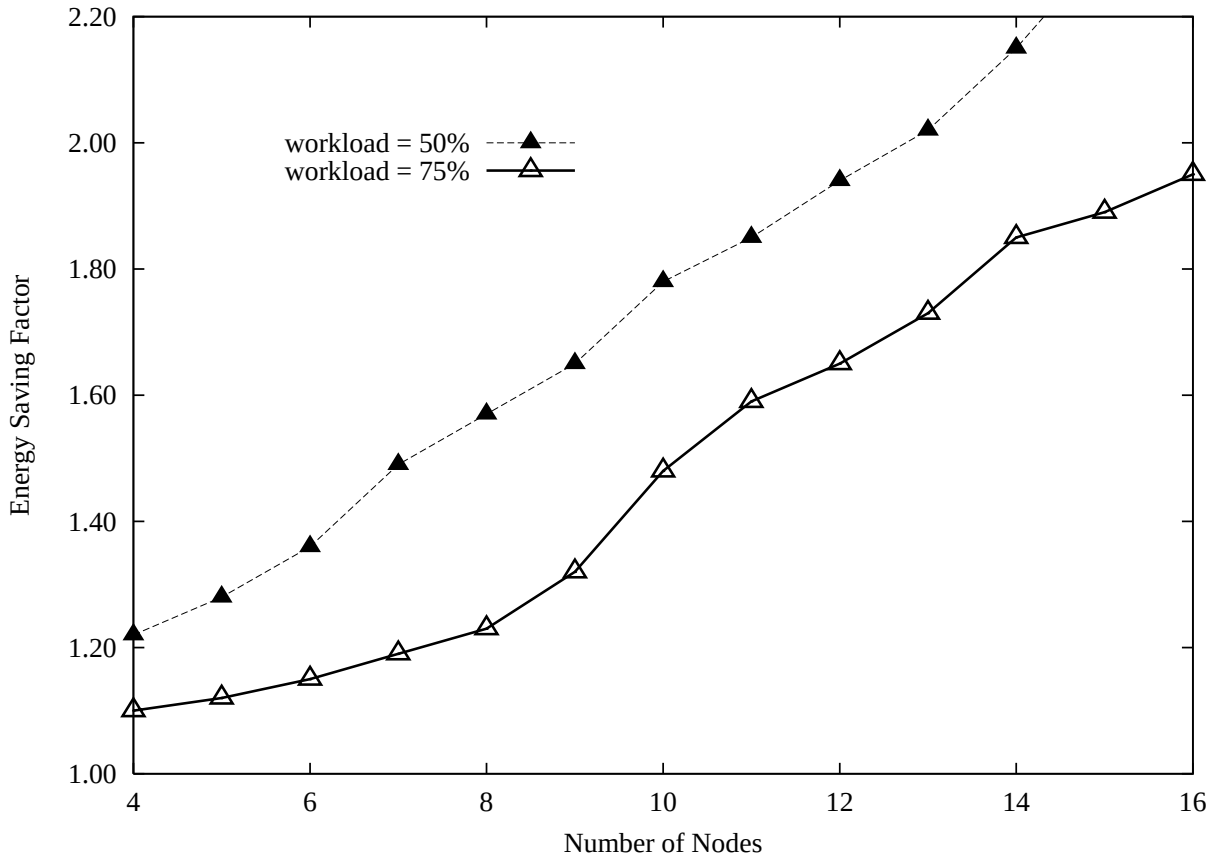


Figure 6.9: Energy saving factor as a function of node density.

Chapter 7

Summary and Future Work

An open real-time system has many advantages over the traditional closed real-time system. When open system is extended to the distributed environment, the flexibility and timing isolation of the hierarchical scheduling scheme can be applied to scheduling network resources. The fact that schedulability of each real-time application can be validated independently is very desirable when the applications are distributed in the network. Any real-time application can be started at run time without having to perform a costly global schedulability analysis. However, scheduling distributed real-time messages in independently developed real-time applications and non-real-time applications is a challenging problem. This dissertation proposes a two-level hierarchical scheme to schedule the applications in a distributed open system. Three different networks with different characteristics are evaluated in the open system framework.

This chapter summaries the results and highlights the contributions presented in the dissertation. We conclude the chapter with suggestions for future research and development in this area.

7.1 Summary of Results

This dissertation proposes a distributed environment and scheduling scheme for independently developed real-time and non-real-time applications. Our open system can handle networks with different characteristics: Myrinet as the high speed cross-bar switch network; Control Area Network being the serial data fieldbus; and Wireless Local Area Network with energy constraint. Further-

more, our distributed open system supports periodic, aperiodic and/or sporadic, and non-real-time messages.

The proposed distributed open system architecture relies on the two-level hierarchical scheduling framework. A communication server implemented as a passive server is used to manage the CPU resource to handle the message processing. Majority of the focus in the dissertation is on scheduling network resources. According to the hierarchical scheduling scheme, all messages from the real-time applications are scheduled hierarchically. The grouping of messages, depending on the network, can be by application, by node, or by message period. Each group of real-time messages is executed by a dedicated server, and all non-real-time applications are executed by a server. The MAC level scheduler then schedules all the servers. When a server is scheduled, it transmits the message at the head of the server ready queue.

The distributed open system has a simple acceptance test. The test makes it possible to start an arbitrary real-time application at run time without having to know the detailed timing attributes of the requesting application and conduct a costly global schedulability analysis.

To evaluate the performance and practicability of the open system on various networks, simulation studies is performed to compare the hierarchical scheduling scheme with the tradition scheduling schemes. Despite the added flexibility, timing isolations, and simplified admission test, the hierarchical scheduling scheme outperforms the closed or traditional network algorithms in network efficiency (and power efficiency in WLAN) at the cost of increased CPU demand. The bandwidth (and energy) constraints in the networks we studied justify that bandwidth is a more scarce resource compare to CPU and hence the approach is considered practical.

To demonstrate the feasibility of the use of passive server to manage network resources in the open system architecture, a prototype communication server is implemented as part of the real-time extension to Windows NT to manage real-time traffics in the Myrinet Cluster.

7.2 Future Work

The two-level hierarchical scheme presented here is for scheduling real-time and non-real-time traffics in a distributed environment, thereby providing bounded end-to-end delay for distributed real-time applications. The communication server design can also be generalized to manage other system resources. The two-level scheme can be easily extended to n-tier depending on the resource being managed.

For each of the networks we studied, many extension and/or enhancements could be investigated. For example, in the case of Myrinet, scheduling real-time messages through multi-hops of myrinet switches is a non-trivial topic. In wireless network, the Scheduled Contention Free Burst still relies on a point coordinator to schedule the bursts, restricting the algorithm to be useful for wireless ad hoc network or wireless sensor network. While open system is designed to support dynamic joining and leaving of applications (or nodes), the MAC layer scheduling scheme may be further enhanced to provide better flexibility. New high-level scheduling algorithms and sufficient schedulability conditions are needed to adapt to lower-level algorithm changes.

References

[1] Z. Deng and J. W.-S. Liu, "Scheduling real-time applications in an open environment," in *Proceedings of Real-Time Systems Symposium*, (San Francisco, California), pp. 308–319, IEEE, December 1997.

[2] Z. Deng, J. W. S. Liu, and J. Sun, "A scheme for scheduling hard real-time application in open sys tem environment," in *Proceedings of 9th Euromicro Workshop on Real-Time Systems*, (Toledo, Spain), pp. 191–199, IEEE, June 1997.

[3] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proc. ACM SIGCOMM*, 1989.

[4] A. Demers, S. Keshav, and S. Shenker, "A generalized processor sharing approach to flow control in integrated services networks: the multiple node case," in *Proc. of IEEE Infocom*, March 1993.

[5] L. Zhang, "Virtualclock: A new traffic control algorithm for packet-switched networks," *ACM Transaction on Computer Systems*, vol. 9, pp. 101–124, 1991.

[6] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Real-Time Systems*, vol. 10, pp. 179–210, 1996.

[7] Z. Deng, J. W. S. Liu, L. Zhang, and S. an d A. Frei, "An open environment for real-time applications," *Real-time Systems*, vol. 16, pp. 155–185, May 1999.

[8] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems," *Real-time Systems Journal*, July 1989.

[9] Z. Deng and J. W. Liu, "Scheduling real-time applications in an open envrionment," in *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 309–319, December 1997.

[10] J. Bennett and H. Zhang, "A generalized processor sharing approach to flow control in integrated services networks: the multiple node case," in *IEEE Transactions on Networking*, vol. 5(5), 1997.

[11] D. Golub, R. Dean, A. Forin, and R. Rashid, "Unix as an application program," in *USENIX Summer Conference*, pp. 87–96, June 1990.

[12] H. Tokuda, T. Nakajima, and P. Rao, "Real-time mach: Toward a predictable real-time system," in *USENIX Mach Workshop*, pp. 73–82, October 1990.

[13] C. W. Mercer, S Savage, and H Tokuda, "Processor capacity reserves: Operating system support for mult imedia applications," in *Proceedings of the IEEE International Conference on Multimedia Com puting and Systems*, IEEE, May 1994.

[14] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, "Resource kernels: A resource-centric approach to real-time and multimedia systems," in *Proc. of SPIE/ACM Conference on Multimedia Computing and Networksing*, January 1998.

[15] G. Lamastra, G. Lipari, G. Buttazzo, A. Casile, and F. Conticelli, "Hartik 3.0: A portable system for developing real-time applications," in *Real-Time Computing Systems and Applications*, October 1997.

[16] A. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," pp. 75–84, December 2001.

[17] C. Seitz, "Myrinet: A gigabit-per-second local-area-network," in *Proc. of IEEE Symposium on Hot Interconnects*, 1994.

[18] M. G. H. Katenvenis, "Fast switching and air control of congested flow in broadband networks," *IEEE Journal on Selected Areas in Communications*, vol. 5, pp. 1315–1326, October 1987.

[19] C. R. Kalmanek, I. Kanakia, and S. Keshav, "Rate controlled servers for very high speed networks," in *Proc. of IEEE Global Telecommunications Conference*, pp. 31–39, December 1990.

[20] I. Philp, *Scheduling Real-Time Messages in Packet-Switched Networks*. PhD thesis, University of Illinois at Urbana-Champaign, 1996.

[21] T. Inukai, "An efficient ss/tdma time slot assignment algorithm," *IEEE Transaction on Communications*, vol. 27, no. 10, pp. 1449–1455, 1979.

[22] W. T. Chen, P. R. Shen, and J. H. Yu, "Time slot assignment in tdm multicast switched systems," *IEEE Transaction on Communications*, vol. 42, no. 1, pp. 149–164, 1994.

[23] Y. K. Tham, "On fast algorithms for tdma switching assignment in terrestrial and satellite networks," *IEEE Transaction on Communications*, vol. 43, no. 8, pp. 2399–2404, 1995.

[24] CiA:, *CANopen Communication Profile for Industrial Systems*. Oct. 1996.

[25] K. Tindell and A. Burns, *Guaranteed Message Latencies for Distributed Safety-Critical Hard Real-Time Control Networks*. Tech. Report YCS229, University of York, May 1994.

[26] K. M. Zuberi and K. G. Shin, "Non-preemptive scheduling of message on the controller area network for real-time applications," in *Proceedings of Real-Time Technology and Applications Symposium*, pp. 240–249, May 1995.

[27] L. Sha, R. Rajkumar, and J. Lehoczky, "Real-time scheduling support in futurebus+," in *Proc. IEEE*, 1990.

[28] J. P. Lehoczky and L. Sha, "Performance of real-time bus scheduling algorithms," *ACM Performance Evaluation Review, Special Issue*, vol. 14, May 1986.

[29] M. D. Natale, "Scheduling the can bus with earliest deadline techniques," in *Proceedings of the Real-Time Systemp Symposium*, December 2000.

[30] S. Baruah, R. Howell, and L. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor," *Real-Time Systems*, vol. 2, 1990.

[31] J. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced aperiodic scheduling in hard-real-time environments," in *Proceedings of Real-Time Systems Symposium*, pp. 261–270, IEEE, December 1987.

[32] R. I. Davis, K. W. Tindell, and A. Burns, "Scheduling slack time in fixed-priority pre-emptive systems," in *Proceedings of Real-Time Systems Symposium*, pp. 222–231, IEEE, December 1993.

[33] S. Ramos-Thuel and J. P. Lehoczky, "On-line scheduling of hard deadline aperiodic tasks in fixed-priority systems," in *Proceedings of Real-Time Systems Symposium*, pp. 160–171, IEEE, December 1993.

[34] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard real-time systems," *Real-Time Systems Journal*, vol. 1, pp. 27–60, June 1989.

[35] T. M. Ghazalie and T. P. Baker, "Aperiodic servers in a deadline scheduling environment," *Real-Time Systems Journal*, vol. 9, pp. 31–67, July 1995.

[36] S. Lu, V. Bharghavan, and R. Srikant, “Fair scheduling in wireless packet network,” in *Proceedings of the ACM SIGCOMM*, pp. 63–74, 1997.

[37] WaveMODEM 2.4GHz Data Manual, *Release 2*. AT&T, 1975.

[38] ITU-T, *Video Coding for Low Bitrate Communication - Draft ITU-T Recommendation H.263*. International Standard, 1996.

[39] D. Seto, J. Lehoczky, L. Sha, and K. Shin, “On task schedulability in real-time control system,” in *Proceedings of the IEEE Real-Time Systems Symposium*, December 1996.

[40] G. Guttazzo, G. Lipari, and L. Abeni, “Elastic task model for adaptive rate control,” in *Proceedings of the IEEE Real-Time Systems Symposium*, (Madrid, Spain), December 1998.

[41] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in hard real time environment,” *Journal of the ACM*, pp. 40–61, 1973.

[42] J. W. Liu, *Real-Time Systems*, ch. 6. Prentice Hall, 2000.

[43] A. Lindgren, A. Almquist, and O. Schelén, “Evaluation of quality of service schemes for ieee 802.11 wireless lans,” in *Proc. of the 26th Annual IEEE Conference on Local Computer Networks*, (Tampa, FL), November 2001.

Vita

Lynn Yuwen Zhang was born in Zigong, China on August 10, 1976. She received her B.S. degree in Computer Science in 1996 from University of Illinois at Urbana-Champaign (UIUC). She worked at the National Center of Supercomputing Applications (NCSA) as research programmer. In 1997, she joined the Real-Time System Laboratory at UIUC and started her MS/PhD program in the Department of Computer Science. She defended her Ph.D. dissertation and joined Intel as a senior automation system engineer in May 2003.