REAL-TIME FAULT-TOLERANT COMMUNICATION IN COMPUTER NETWORKS

by

Qin Zheng

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering: Systems)
in The University of Michigan
1993

Doctoral Committee:

Professor Kang G. Shin, Chairman
Professor Semyon M. Meerkov
Professor John F. Meyer
Professor Katta G. Murty
Assistant Professor Chinya V. Ravishankar
Associate Professor Wayne E. Stark

ABSTRACT

REAL-TIME FAULT-TOLERANT COMMUNICATION IN COMPUTER NETWORKS

by
Qin Zheng

Chair: Kang G. Shin

A network is required to provide the users with a convenient means of guaranteeing delay bounds in message transmission and making message transmission tolerant of network component failures. Solutions to this problem will greatly improve the quality of service of the contemporary computer networks and expand their application domains to such areas as distributed real-time controls and digital continuous-media (motion video, audio) transmissions.

Our solution to the problem is to use a new transfer mode called a *real-time channel* which guarantees the timely delivery of messages like the circuit-switched transmission while preserving the high transmission efficiency of the packet-switched mode. We give a comprehensive coverage of this new transfer mode, from the fundamental deadline scheduling theory and real-time channel protocols, to the detailed hardware implementation. Using the spatial redundancy of a network topology, real-time fault-tolerant communication is achieved by enhancing the basic real-time channels to be Single Failure Immune (SFI) or Isolated Failure Immune (IFI). Backup channels can also be used to increase the reliability of real-time channels.

The issue of establishing real-time channels over shared-medium Local Area Networks (LANs) is then discussed, which is of practical importance since most end systems are connected to a LAN first and then to a point-to-point Wide Area Network (WAN). We also present a technique which can significantly improve the FDDI's capacity of supporting real-time traffic with a few simple modifications to its Medium Access Control (MAC) protocols. Finally, the application of our results for multimedia networking is illustrated through simulations.

ACKNOWLEDGEMENTS

First, I would like to express my deepest gratitude to Prof. Kang G. Shin. As my thesis advisor, he helped me select the right topic at the very beginning which importance to the thesis can never be overemphasized. Throughout the course of this work, he has given me constant encouragement, generous support, insightful comments, and invaluable suggestions which benefited not only the completion of this thesis but also my career in a long time to come. As many of his students have pointed out, it would be hard to find a person who is more concerned about his students and one could not have a more dedicated and more caring advisor.

I would also like to express my appreciation to other members of the thesis committee, Profs. Semyon Meerkov, John Meyer, Katta Murty, Chinya Ravishankar, and Wayne Stark for their time and constructive comments on this thesis.

I have benefited greatly from many other people. In particular, the design of a fast deadline scheduler presented in Chapter 4 was a collaborative project done with Atri Inderesan under the guidance of Prof. S. Abraham, as well as Prof. Kang G. Shin. Part of the simulation work on multimedia networking in Chapter 8 was done with Emmanuel Abram-Profeta. Parameswaran Ramanathan and his student Moncef Hamdaoui pointed out an error in a previous version of Theorem 2.6. Also, without the early work on real-time channels by Domenico Ferrari, Dinesh C. Verma, and Dilip D. Kandlur, this thesis would not have been possible.

Thanks also to the Department of EECS, the Office of Naval Research, and the National Science Foundation for providing financial support during the course of my graduate program.

My final acknowledgments go to my wife, my parents, and my parents–in–law for their dedicated support and understanding.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

CHAPTER 1

INTRODUCTION

1.1 Background

Today's communication networks are characterized by specialization [1]. Different networks provide different communication services. Some of the most important existing communication networks include

1. a telephone network which offers the customers two-way voice conversations,
2. a computer network which transports computer data traffic, and
3. a cable TV (CATV) network which delivers television signals.

Each of these networks was designed specially for a specific service and is often not at all applicable to deliver different services. For example, a CATV network does not support voice conversations, and a telephone network does not deliver TV signals. Transmission of voice over a computer network is very problematic because of the unpredictable end–to–end delays, and transmission of computer data over the telephone network is very inefficient due to the low transmission speed achievable.

An important consequence of this service specialization is the existence of a large number of independent networks, each requiring its own design phase, manufacturing and maintenance. In addition, the capacity of each network must be designed for every individual service type. Even if resources are freely available in one network, they cannot be used by another service type. For instance, the peak hours in a telephone network are between 9 a.m. and 5 p.m., whereas the peak hours in a CATV network are during evening. Since resource sharing is impossible, each network must be dimensioned for the peak hour traffic.

It is thus desirable to integrate a large number of different networks into a single universal network which is capable of providing all types of services. A single service–independent network will have the following advantages [1]:

1. Advances in technology may surpass the bandwidth used of existing communication services. New applications may require new communication services. A network capable of delivering all types of services will be able to adapt itself to changing or new needs.
2. Efficient sharing of available resources for all services.
3. Since only one network needs to be designed, manufactured and maintained, the overall cost of network design, manufacturing, operations and maintenance will be lower.

The work on integrating different networks into a single universal network has been done in the following two areas:

1. Telephone networks are made to support bursty computer data traffic and high rate video signals. The resultant network is called the Broadband Integrated Service Digital Network (BISDN). The key technology to be used for BISDNs is the Asynchronous Transfer Mode (ATM) which dynamically multiplexes packets (or cells in the ATM terminology) over transmission links. Compared with the Synchronous Transfer Mode (STM) which is used in current telephone networks, ATM is more suitable for handling varying-rate signals since it uses transmission resources more flexibly and efficiently.

2. Computer networks are made capable of supporting real–time traffic which requires the timely delivery of messages. Examples of real–time traffic include digital voice/video streams and control/monitoring signals in distributed control/manufacturing systems. The key problem with real–time communication is to improve the quality of service of today's computer networks such that not only can bursty computer data be efficiently transported, but also real–time traffic can be satisfactorily handled.

Traditionally, the first area has been investigated by telecommunication engineers, while computer scientists have been working in the second area. However, the boundary between two areas is fast becoming unclear, and the results in both areas are converging to a single technology with which a universal network will be built.

This thesis deals with real–time communication in computer networks. The approaches developed in this thesis for real–time communication bear many similarities to the ATM — a key technology to be used for the BISDN.

1.2 Research Objective

Advances in high–speed computer networking during the past decade have made available a wide range of network services, such as *email* to send electronic messages worldwide, *ftp* to transfer files between computers, and *telnet* for remote login to another machine. However, the contemporary computer networks fail to provide adequate *quality of service*. A message between two sites could get delayed for an unpredictably long period or even lost due to network congestion and/or component failures.

Insufficient service quality limits the application domain of computer networks to non real–time data communication only. Services which are not widely provided include digital continuous–media (motion video, audio), timely delivery of urgent messages in real–time control/manufacturing systems, and fast request–reply communication [2].

The objective of this research is thus to study how the quality of service can be improved in computer networks. Specifically, in addition to non real–time data communication, we want to provide network users with services which guarantee the *timely* and *reliable* delivery of messages. Achievement of this objective will make computer networks capable of supporting many applications including those mentioned above and those which may not be known at present.

1.3 Approach

A typical computer network is hierarchically organized. End systems in one or several buildings/floors are connected together by a local area network (LAN) like the

Ethernet or IBM token ring. Several LANs are combined into a campus area network via a high–speed backbone network like the FDDI token ring. These campus area networks are then linked together via long distance lines to form the current Internet.

The Internet provides users with two basic types of communication service: datagrams and virtual circuits. Typical examples of these two types are the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). The datagram service provides connectionless, unreliable communications between two end systems where each data unit (datagram) is sent independently and there is no guarantee that the datagrams will ever get delivered or delivered correctly. The virtual circuit service, on the other hand, provides connection–oriented, reliable communications, and guarantees all messages to get delivered correctly and in sequence. Each type of service has its own application domains. Datagrams are used for short and/or urgent communications because no connection establishment procedure is needed, while virtual circuits are more suitable for those applications requiring reliable and sequenced delivery of messages.

An important feature that both datagrams and virtual circuits do not support is the guaranteed *timely* delivery of messages despite numerous applications — such as interactive voice/video communications, time–constrained remote operations, and real–time control/monitoring — that require messages to be delivered within pre-specified delay bounds. For these applications, we need a third type of service which guarantees the timely, sequenced delivery of messages.

The need for providing services which *guarantee* the timely delivery of messages in computer networks was first noted in [3], recognizing that in addition to the TCP, at least two other types of protocols are desirable: (i) a protocol guaranteeing sequenced, timely delivery of messages without considering the reliability of message delivery, and (ii) a protocol guaranteeing both timeliness and high reliability. The first protocol is to be used for real–time voice/video transmission, and the second protocol could be used for distributed control/manufacturing systems. However, little work has been done in this area.

An easy way to implement real–time communication would be to use the circuit–switched transmission. Given the user's maximum traffic generation rate, one can use a dedicated circuit between two end systems with an adequate bandwidth to guarantee the timely delivery of all messages. If the bandwidth of a link is greater than that a single channel requires, several channels can be established through the link using either *frequency–division multiplexing* (FDM) or *time–division multiplexing* (TDM) techniques, as is usually done for telecommunication systems. These multiplexing techniques, however, do not exploit the bursty nature of data traffic (resulting in inefficient use of link capacity) and the different requirements of traffic types (resulting in an inflexible allocation of link bandwidths), and thus are rarely used in computer networks.

The packet–switched transmission, on the other hand, uses the link capacity more efficiently and flexibly by dynamically allocating link bandwidths according to the traffic demands. However, due to the queueing delays at transmission links (e.g., when more than one packet need to be transmitted over a link, only one can be transmitted immediately and all others would have to wait), message delivery times are not predictable.

To counter this problem, a new transfer mode, called the *real–time channel*, has recently been proposed to support real–time communication in packet–switched networks [4]. Real–time channels use two techniques to guarantee end–to–end message delay bound: admission control of channels and deadline scheduling of packet transmissions.

Like the circuit–switched transmission, admission control requires a process requesting real–time service to establish a connection (i.e., a real–time channel) before starting

packet transmissions. Non real–time messages are transmitted in an ordinary way without needing a channel establishment phase. However, in order not to affect the quality of service guaranteed to real–time channels by the unpredictable non real–time traffic, non real–time messages are given lower transmission priority than that of real–time messages transmitted via real–time channels.

A real–time channel establishment request may be accepted or rejected, depending on the current network–load condition. Admission control is necessary because message delay bounds cannot be guaranteed without controlling the traffic load of the network. If a channel request is rejected, the requesting process has the following three options,

1. Wait for a period of time and request the establishment of the channel again. Since real–time channels are established or removed from time to time, a channel which cannot be established at one time may be establishable at some other time. This is usually used for critical real–time applications like in a distributed control/manufacturing system where an application should not be started before adequate communication channels are established.

2. Reduce the quality of service requirement, e.g., increase the requested end–to–end delay bound, and try to establish the channel again. This is sometimes useful for non critical real–time applications like interactive voice/video transmissions. When the first channel establishment failed, the user may be willing to accept a connection with a lower quality than the previously requested.

3. Transmit messages as non real–time ones without establishing a real–time channel. In this case, there will be no guarantee on the message delivery delay bound. This method is suitable for randomly generated urgent messages for which transmission without guarantee is better than holding them at the source node until the delivering-time guarantee can be arranged.

Each message is split into packets which may be all of the same size (like in an ATM network) or have different sizes (like in the current computer networks). Packet transmissions are scheduled as follows. Real–time packets are given higher transmission priority than non real–time packets. Each real–time packet is assigned a deadline over each link it traverses which is calculated from the requested end–to–end delay bound and the generation time of the packet. When several real–time packets contend for use of the same link, the packet with the earliest deadline is transmitted first. There are two advantages of using the deadline scheduling.

1. Minimal effects of queueing delays: For real–time applications, we need to control the *maximum* packet delay. Unlike the average delay, the queueing delays at transmission links have significant effects on the maximum packet delays, even when the traffic load is not heavy. The deadline scheduling policy can minimize the effects of queueing delays in the sense that given a set of packets with deadlines, if they are schedulable under any scheduling policy (i.e., every packet can be transmitted before its deadline), so can they under the deadline scheduling policy [5]. Thus, the deadline scheduling policy gives the communication network more capacity to accommodate real–time channels than that with other scheduling policies. In other words, the deadline scheduling reduces the probability of rejecting channel establishment requests.

2. Channel protection: When establishing a new real–time channel, the worst–case queueing delay is calculated for each link under the assumption that all existing real–time channels over this link will generate messages according to their pre-specified patterns (i.e., do not exceed their *a priori* specified maximum message generation rate and maximum message length). In practice, however, it is possible that some channels exceed their pre-specified limits. By properly assigning deadlines to the packets over each transmission link they traverse, the deadline scheduling policy can ensure that those channels exceeding their pre-specified limits will not affect the timely delivery of the other channels' packets. Other scheduling policies, like the First–In–First–Out (FIFO), Exhaustive Round Robin or Priority Scheduling, do not possess this property.

In summary, real–time channels established in a packet–switched network guarantee the timely delivery of messages like dedicated circuits in a circuit–switched network. Each real–time channel guarantees the end–to–end delay bound as long as the source node does not exceed the pre-specified limits of message length and generation rate. However, unlike a dedicated circuit in a circuit–switched network, a real–time channel does not reserve any transmission bandwidth. The links are free to transmit other packets whenever there are no real–time packets to be transmitted, thus allowing efficient use of network bandwidth.

It is worth emphasizing that only those users who need guaranteed timely delivery of their messages would use real–time channels. Non real–time data services provided by today's computer networks are still available to all other users who do not have tight delay requirements to meet. In other words, using real–time channels is a way to *enhance* computer networks with one more service type for users to choose from.

Research on real–time channels is still in its infancy. This thesis addresses some of the key issues of the real–time channel. Specifically, we will derive real–time channel establishment conditions, specify a detailed real–time channel approach, investigate the implementation issues, and enhance real–time channels to be fault–tolerant. Establishment of real–time channels over local area networks and the application of real–time channels for multimedia networking are also addressed.

1.4 Organization of the Thesis

Since real–time channels are connection–oriented, one must set up a connection before sending messages. A fundamental problem is then to find the conditions under which a new channel request can be granted without violating existing guarantees. Previously known conditions either used some restrictive assumptions on the channel parameters [4] or were sufficient but not necessary [6]. Under such conditions, a channel establishment request could be rejected even if the network has the capacity to accommodate it.

We derive in Chapter 2 some results of deadline scheduling theory from which sufficient and necessary real–time channel establishment conditions can be obtained without any assumptions on the channel parameters [7]. Thus, under these conditions, the network transmission resources can be fully utilized when real–time channels are established.

Based on the deadline scheduling theory obtained in Chapter 2, we present in Chapter 3 the detailed procedures for channel establishment and message transmissions. These results are the first detailed specification of transport, network, and data link layer protocols of real–time channels. They also show the feasibility of the real–time channel concept.

One special requirement of the real–time channel protocol is that the deadline scheduling policy be used which specifies that if more than one packet contend for the same transmission link, the one with the earliest deadline should be transmitted first. The deadline scheduling of packet transmissions can be implemented in two ways:

1. Incoming packets are always placed at the end of a waiting queue. When the transmitter finishes transmitting a packet, it searches the waiting queue and picks up the packet with the earliest deadline to transmit.

2. The waiting queue is organized such that the packet with the earliest deadline is always placed at the head of the queue. Thus the transmitter always takes the packet from the head of the queue to transmit. However, when a new packet arrives, the queue must be searched for a proper position to insert the packet.

The time needed to find a proper packet to transmit or a proper position to insert a new packet is called the *scheduling time*. Unlike the commonly–used FIFO scheduling, the time needed for deadline scheduling is not negligible and dependent on the length of the waiting queue. To ensure the success of real–time channels, it is crucial to keep the maximum packet scheduling time below the time needed to transmit one packet, which is a challenging problem for high-speed networks.

In Chapter 4, we present a design of hardware deadline scheduler which can schedule one packet in at most 12 clock cycles [8, 9]. In other words, the scheduler can schedule n million packets per second with a $12 \times n$ MHz clock frequency. The scheduling time is independent of the queue length as long as the total number of real–time channels passing through a link does not exceed the designed capacity. The design of the hardware deadline scheduler is verified and simulated with the hardware description language VERILOG.

The reliability problem of real–time channels is investigated in Chapter 5. A basic real–time channel is susceptible to component failures since all messages of the channel are transmitted along a pre-determined path. A single node/link failure in the path could disable the whole channel. In networks with sufficient connectivity like meshes and cubes, a basic real–time channel can be made Single Failure Immune (SFI) by adopting a semi-dynamic routing strategy and adding some extra links/nodes to the channel [10]. An SFI real–time channel guarantees the timely delivery of messages as long as there is no more than one link/node failure on the channel.

We also show how a class of reliable real–time channels, called Isolated Failure Immune (IFI) real–time channels, can be established in hexagonal mesh networks [11, 12]. An IFI real–time channel can tolerate any node/link failures which are not adjacent.

For networks where the SFI real–time channels can not be established, backup real–time channels can be established and switched in quickly in case a primary channel becomes faulty.

Chapter 6 discusses real–time communication in shared–medium local area networks (LANs). This issue is important since end systems are first connected via LANs. An end–to–end delay bound can not be guaranteed without first achieving real–time communication in LANs. This chapter shows how the real–time channels can be established over FDDI local area networks. A modification to the FDDI, called the FDDI-M, is also proposed which significantly improves a network's capacity of supporting real–time traffic [13, 14]. The new protocol can be implemented with minor modifications to the existing FDDI networks.

One of the main applications of real–time communication is multimedia networking. Interactive continuous-media communication like digital motion video needs timely delivery of messages. Messages arrived late will be discarded but deteriorate transmission quality. Conventional circuit–switched transmission is not suitable since the signal rate in multimedia applications usually varies greatly. Usage of the fixed bandwidth circuits for those applications with highly–varying signal rates wastes link bandwidth.

In Chapter 7, we show how the real–time channel protocol can be used to support the transmission of digital motion video [15]. When comparing to the conventional circuit switching, the real–time channel protocol is shown to remove the waste of link bandwidth, and when comparing to the conventional packet switching, the real–time channel protocol is shown to remove the uncertainty in message transmission delays. In shared–medium local area networks, the newly proposed FDDI-M is also shown to outperform the existing FDDI and FDDI-II protocols.

The thesis concludes with Chapter 8 which summarizes our contributions and suggests future work.

The primary application of the results in this thesis is computer networks. However, the results are generic in nature and applicable to other areas like message–passing multicomputers and telecommunication systems.

CHAPTER 2

DEADLINE SCHEDULING THEORY

2.1 Introduction

As discussed in Chapter 1, real–time channels use deadline scheduling for packet transmissions. The advantages of using the deadline scheduling policy are the minimal effects of contention delays and channel protection. However, as pointed out in [6], one major problem of using the deadline scheduling is the difficulty in computing guarantees. Especially, there are no known solutions to the *schedulability problem*: given a set of real–time channels, can all messages of these channels be delivered before their requested deadlines? It is essential to solve the schedulability problem if the deadline scheduling policy is to be used for real–time channels.

Deadline scheduling theory was first studied in [5] which presented some results on periodic task scheduling on a single processor. A schedulability condition was obtained under the condition that a task's deadline always equals the generation time of the next task. However, this result is hardly useful for real–time channels since a message's deadline over one link does not equal to the generation time of the next message in most cases.

Ferrari and Verma [4] obtained a solution to the schedulability problem under an assumption that the summation of the maximum message transmission times over all real–time channels passing through a link is not larger than the minimum message interarrival times of these channels. This assumption is also quite restrictive in practice since it limits the traffic types to be serviced. Without using this assumption, Kandlur *et al.* [6] established a *sufficient* condition to check the schedulability of channels. They first derived the schedulability conditions from a priority scheduling policy which assigns different priorities to the packets of different channels and the packet with the highest priority in the waiting queue is always transmitted first. Since a set of channels which are schedulable under the priority scheduling policy are also schedulable under the deadline scheduling policy, this condition is a sufficient schedulability condition for the deadline scheduling policy.

It can be proved that under the assumption of [4], the sufficient condition of [6] is equivalent to the condition of [4]. So, the result of [6] subsumes that of [4]; that is, [6] can deal with situations where the assumption of [4] fails to hold. However, using sufficient schedulability conditions for establishing real–time channels may still under–utilize the network's transmission capacity since a violation of the sufficient conditions does not necessarily mean that the channels cannot be established.

The goal of this chapter is thus to obtain "true" schedulability conditions that are both necessary and sufficient without any assumption about the traffic types to be serviced. Using these conditions, the network's transmission capacity can be fully utilized to accommodate real–time channels.

This chapter is organized as follows. Section 2.2 gives a real–time channel model based on which the schedulability problem and minimum delay bound problem are defined. The solutions to the two problems under the preemptive and non-preemptive deadline scheduling policies are derived in Sections 2.3 and 2.4, respectively. The chapter concludes with Section 2.5.

2.2 Real–time Channel Model

A real–time channel is uni-directional. A bi-directional real–time channel can be created by setting up two separate uni-directional channels. The reason for this is that the traffic patterns of both directions could be significantly different. So it is efficient and convenient to consider one direction at a time.

To set up a real–time channel, the requesting process must specify its traffic generation pattern. Because of the limited capacity of transmission links, no bounded message delivery delay can be guaranteed without this information. We use two parameters, T and M, to describe a traffic pattern, where T is the *minimum* message inter-generation time and M is the *maximum* message size. It is reasonable to assume prior knowledge of these parameters for many applications, such as interactive voice/video transmission and real–time control/monitoring. In other applications where the traffic pattern is less predictable, the estimated values of T and M could be used. A process may exceed its pre-specified maximum message size and/or message generation rate at the risk that these messages may not be delivered within the pre-specified delay bound. But the real–time channel protocol will ensure that this particular process will not affect the guarantees to other channels.

The requesting process also specifies the addresses of the source and destination nodes of the channel, and an end–to–end delay bound D within which this process requires all its messages be delivered. Together with the the traffic generation pattern, a real–time channel can thus be defined as follows.

Definition 2.1 (Real–time channel) .
A real–time channel, described by a tuple (T, M, D), is a connection between two nodes which guarantees that every message generated at the source will be delivered to the destination in a time period no longer than D under the conditions that (C1) the message inter-generation time is not smaller than T, and (C2) the message size is not larger than M.

From this definition, we see that a real–time channel is a very convenient way to achieve real–time communication. In a network which provides real–time channel services, users can set up channels with adequate bandwidth (i.e., T and M) and delay bound (i.e., D) suited to their applications. This flexibility in establishing different types of channels helps the better management of network resources and reduces the costs to users if they are charged for their connections. For example, a user will pay lower connection fee for a voice channel than a video channel since the former needs less bandwidth. Also, a channel with larger requesting delay bound D costs less than the one with a smaller D. This is in sharp contrast to the conventional circuit–switched transmission where users do not have many choices on the bandwidth and quality (e.g., the delay bound D) of the circuits.

A fundamental problem to the establishment of real–time channels is to find the conditions to check whether the requested channel can be established. In this chapter, we derive channel establishment conditions over one link. In other words, we assume that each real–time channel is composed of one link only. We also assume that C1 and C2 in

Definition 2.1 are always satisfied for each real–time channel. The results will be generalized in the next chapter for the establishment of multi-link real–time channels in an environment where some channels may violate the assumptions C1 and C2.

For the convenience of the following discussions, we use another variable C to denote the maximum message transmission time of a real–time channel. In other words, if R is the link transmission rate, then $C = M/R$. The *delay* of a message over a link is defined as the time between the generation of the message and the transmission of the last bit of the message. The deadline of a message belong to a real–time channel (T, C, D) is calculated as its generation time plus the requested delay bound D. Then the *schedulability* of a set of channels over a link is defined as follows.

Definition 2.2 (Channel schedulability) .
A set of real–time channels $\tau_i = (T_i, C_i, D_i)$, $i = 1, 2, ..., n$, is said to be schedulable *over a link if for all $1 \leq i \leq n$, the maximum delay experienced by channel i's messages over the link is not larger than the requested delay bound D_i.*

To establish real–time channels, we need solutions to the following two problems.

Problem 2.1 (Channel schedulability) .
Given a set of real–time channels $\tau_i = (T_i, C_i, D_i)$, $i = 1, 2, ..., n$, are they schedulable over a link?

Problem 2.2 (Minimum delay bound) .
Suppose $n - 1$ channels, $\tau_i = (T_i, C_i, D_i)$, $i = 1, 2, ..., n - 1$, are schedulable over a link. Given a new channel τ_n with the minimum message inter–arrival time T_n and the maximum message transmission time C_n, what is the minimum value of D_n such that all n channels are still schedulable over the link?

We derive the solutions to Problems 2.1 and 2.2 in the rest of this chapter, which form the fundamental deadline scheduling theory for real–time channels.

2.3 Preemptive Deadline Scheduling

Deadline scheduling of message transmissions can be done in two ways: preemptive and non-preemptive. With a preemptive scheduling policy, the transmission priority is always given to the message with the earliest deadline. When a message with an earlier deadline is generated, the transmission of the current message is preempted and will be resumed after all messages with earlier deadlines are transmitted. The time needed for the preemption and resumption of message transmissions is assumed to be negligible. With a non-preemptive scheduling policy, messages are divided into packets and the preemption of a message transmission is allowed only at the boundaries of the packets. We derive solutions to Problems 2.1 and 2.2 under the preemptive scheduling policy in this section. The results will be extended to the non-preemptive scheduling policy in the next section.

The channel schedulability conditions are given in the following theorem.

Theorem 2.1 (Solution to Problem 2.1) .
A set of n channels $\tau_i = (T_i, C_i, D_i)$, $i = 1, 2, ..., n$, are schedulable over a link under the preemptive deadline scheduling policy if and only if

$$\forall t \geq 0, \quad \sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i \leq t,$$

where $\lceil x \rceil^+ = n$ *if* $n-1 \leq x < n$, $n = 1, 2, ...,$ *and* $\lceil x \rceil^+ = 0$ *for* $x < 0$.

Proof of the necessary condition: Let 0 be the starting time for the system. In other words, the system is empty (of messages) at time $t = 0$. Then, $\forall t > 0$, a necessary condition for no messages to miss their deadlines in a time period $[0, t]$ is that the amount of time, T, needed to transmit all those messages generated during $[0, t]$ with deadlines $\leq t$ is not greater than t. Since the minimal message inter–generation time of channel i is T_i, there are at most $\lceil (t - D_i)/T_i \rceil^+$ messages generated by channel τ_i during $[0, t]$ with deadlines $\leq t$, which need at most $\lceil (t - D_i)/T_i \rceil^+ C_i$ units of time to transmit. Thus, the maximum value of T is $\sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i$. This proves the necessary condition.

Proof of the sufficient condition: We prove this by contradiction. Suppose a message misses its deadline at time t_1, meaning that at least one message with deadline $\leq t_1$ has *not* been transmitted over the link by t_1 (a message is said to have been transmitted when the last bit of the packet leaves the transmitting node). Then, from the property of the preemptive deadline scheduling policy, there must exist $t' < t_1$ such that during the time period $[t', t_1]$, the node is busy transmitting only those messages with deadlines $\leq t_1$. Let t_0 be the smallest such t', then there are no messages with deadlines $\leq t_1$ queued at the link at time t_0^-. Thus, it is concluded that in the time period $[t_0, t_1]$, the link is busy transmitting only those messages which are generated at the link during the time period $[t_0, t_1]$ and having deadlines $\leq t_1$. Based on the same reasoning as the proof of the necessary condition, the maximum amount of time needed to transmit these messages is $T = \sum_{i=1}^{n} \lceil (t_1 - t_0 - D_i)/T_i \rceil^+ C_i$. Since one message misses its deadline at t_1, this T must be larger than $t_1 - t_0$, that is,

$$\sum_{i=1}^{n} \lceil (t_1 - t_0 - D_i)/T_i \rceil^+ C_i > t_1 - t_0.$$

By letting $t = t_1 - t_0$, the above inequality contradicts the condition that $\forall t \geq 0$, $\sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i \leq t$. □

An interesting special case of Theorem 2.1 is when $D_i = T_i$ for all $1 \leq i \leq n$. Since $\lceil (t - T_i)/T_i \rceil^+ \leq t/T_i$, the inequality of Theorem 2.1 is satisfied if the maximum utilization of the link, $\sum_{i=1}^{n} C_i/T_i$, is not greater than 1. Also, it is easy to see that the channels are not schedulable if the maximum utilization of the link exceeds 1. Thus, it can be concluded that when the requested delay bounds are equal to the minimum message inter–arrival times, the channels are schedulable over the link if and only if the maximum link utilization does not exceed 1. This is the well–known result obtained in [5] for the problem of scheduling periodic tasks over a single processor. Thus Theorem 2.1 is a generalization of the schedulability condition of [5].

Two additional properties about the channel schedulability over a single link follow immediately from Theorem 2.1.

1. Increasing the requested delay bounds D_i's will not affect the schedulability of channels.

2. A new channel can always be added if its requested delay bound is large enough and the total utilization of the link does not exceed 1.

These two properties are stated formally in the following corollary.

Corollary 2.1 (Channel schedulability properties) .

(1) *Suppose a set of channels* $\tau_i = (T_i, C_i, D_i)$, $i = 1, ..., n$, *are schedulable over a link, then for any* $D_i' \geq D_i$, *the set of channels* $\tau_i' = (T_i, C_i, D_i')$, $i = 1, ..., n$, *are also schedulable over the link.*

(2) *Suppose* $n-1$ *channels* $\tau_i = (T_i, C_i, D_i)$, $i = 1, ..., n-1$, *are schedulable over a link, then by adding one more channel* $\tau_n = (T_n, C_n, D_n)$, *the set of* n *channels* $\tau_i = (T_i, C_i, D_i)$, $i = 1, ..., n$, *are also schedulable over the link if the maximum link utilization* $\sum_{i=1}^{n} C_i/T_i < 1$ *and* $D_n \geq T_n + t_n$, *where* $t_n = \max\{D_1, ..., D_{n-1}, (\sum_{i=1}^{n-1}(1 - D_i/T_i)C_i + C_n)/(1 - \sum_{i=1}^{n-1} C_i/T_i))\}$.

Proof: (1) is a direct result of Theorem 2.1.

To prove (2), using Theorem 2.1 we need to show that $\forall t \geq 0$, $\sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i \leq t$. For $t < D_n$, $\sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i = \sum_{i=1}^{n-1} \lceil (t - D_i)/T_i \rceil^+ C_i$. Since the first $n-1$ channels are schedulable over the link, from Theorem 2.1, the right-hand side of the above equation can not be larger than t. Thus, $\sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i \leq t$ for $t < D_n$. For $t \geq D_n$,

$$\sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i \leq \sum_{i=1}^{n-1} (1 - D_i/T_i)C_i + (\sum_{i=1}^{n} C_i/T_i)t + (1 - (T_n + t_n)/T_n)C_n.$$

From the definition of t_n, we have

$$\sum_{i=1}^{n-1} (1 - D_i/T_i)C_i \leq (1 - \sum_{i=1}^{n-1} C_i/T_i)t_n.$$

Thus,

$$\sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i \leq t_n + (\sum_{i=1}^{n} C_i/T_i)(t - t_n).$$

Since $\sum_{i=1}^{n} C_i/T_i < 1$, $\sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i \leq t$ follows. □

Theorem 2.1 has a neat mathematical form and is useful for deriving properties about the channel schedulability. However, one may find it difficult to use for solving the schedulability problem (Problem 2.1) in practice, because the inequality of Theorem 2.1 is supposed to be checked over an infinite length interval $[0, \infty)$. This problem can be solved with the following two observations. First, the left-hand side of the inequality is a piece-wise constant function. Thus, we only need to check the inequality at some discrete points in $[0, \infty)$. Second, there exists a point t_{max} such that under the condition that the total utilization of the link $\sum_{i=1}^{n} C_n/T_n < 1$, the inequality of Theorem 2.1 always holds for $\forall\ t \geq t_{max}$. So, we only need to consider a *finite* set of points to validate the inequality of Theorem 2.1. To this end, we have the following theorem which is a practically-realizable version of Theorem 2.1.

Theorem 2.2 (Practical solution to Problem 2.1) .
A set of channels $\tau_i = (T_i, C_i, D_i)$, $i = 1, 2, ..., n$, *are schedulable over a link with the preemptive deadline scheduling policy if and only if both of the following hold:*

1. $\sum_{i=1}^{n} C_i/T_i \leq 1$.

2. $\forall t \in S,\ \sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i \leq t,$
where $S = \cup_{i=1}^{n} S_i,\ S_i = \{D_i + kT_i :\ k = 0, 1, ..., \lfloor (t_{max} - D_i)/T_i \rfloor\},$
and $t_{max} = \max\{D_1, ..., D_n, (\sum_{i=1}^{n}(1 - D_i/T_i)C_i)/(1 - \sum_{i=1}^{n} C_i/T_i)\}.$

Proof: The first condition is easy to prove since if the maximum link utilization $\sum_{i=1}^{n} C_i/T_i > 1$, the message queueing delay will tend to ∞. Thus, there is no way that these channels can be scheduled over the link.

To prove the second condition, we use Theorem 2.1. Since the value of $\lceil (t - D_i)/T_i \rceil^+ C_i$ changes only on the set $S_i' = \{D_i + kT_i : k = 0, 1, ...\}$, we only need to check the inequality of Theorem 2.1 on the set $S' = \cup_{i=1}^{n} S_i'$.

Furthermore, $\lceil (t - D_i)/T_i \rceil^+ \leq 1 + (t - D_i)/T_i$ for all $t \geq \max\{D_i : i = 1, ..., n\}$. So, it is easy to verify that for $t \geq t_{max}$, the inequality of Theorem 2.1 always holds. So, we only need to check the inequality on the set $S' \cap \{t : t \leq t_{max}\}$, which is the set S. □

The following example shows how Theorem 2.2 can be used to solve the schedulability problem.

Example 2.1 . *Given three channels* $\tau_1 = (T_1, C_1, D_1) = (10, 2, 5)$, $\tau_2 = (T_2, C_2, D_2) = (8, 4, 8)$, *and* $\tau_3 = (T_3, C_3, D_3) = (12, 3, D_3)$. *Use Theorem 2.2 to check their schedulability for* $D_3 = 9$ *and* $D_3 = 8$.

Solution: First we check that the link utilization $\sum_{i=1}^{3} C_i/T_i = 0.95 < 1$.

For $D_3 = 9$, $t_{max} = \max\{D_1, D_2, D_3, (\sum_{i=1}^{3}(1 - D_i/T_i)C_i)/(1 - \sum_{i=1}^{3} C_i/T_i)\} = \max\{5, 8, 9, 35\} = 35$. Then $S_1 = \{5, 15, 25, 35\}$, $S_2 = \{8, 16, 24, 32\}$, $S_3 = \{9, 21, 33\}$, and $S = S_1 \cup S_2 \cup S_3$. It is easy to verify that $\forall t \in S$, $\sum_{i=1}^{3} \lceil (t - D_i)/T_i \rceil^+ C_i \leq t$. Thus, the three channels are schedulable with $D_3 = 9$.

For $D_3 = 8$, $t_{max} = \max\{D_1, D_2, D_3, (\sum_{i=1}^{3}(1 - d_i/T_i)C_i)/(1 - \sum_{i=1}^{3} C_i/T_i)\} = \max\{5, 8, 8, 40\} = 40$. Then $S_1 = \{5, 15, 25, 40\}$, $S_2 = \{8, 16, 24, 32, 40\}$, $S_3 = \{8, 20, 32\}$, and $S = S_1 \cup S_2 \cup S_3$. At $t = 8$, $\sum_{i=1}^{3} \lceil (t - d_i)/T_i \rceil^+ = 9$. Thus, the inequality of Theorem 2.2 is not satisfied. We conclude that the three channels are not schedulable with $D_3 = 8$. □

Using the first result of Corollary 2.1, we see that in the above example, 9 is the minimum integer value of D_3 such that the three channels are schedulable. So, if one wants to establish channel τ_3 over a link on which τ_1 and τ_2 already exist, the minimum requested delay bound he/she can ask is 9. We then want to know if there is an efficient way to find this minimum requested delay bound. This is the minimum delay bound problem (Problem 2.2) defined in Section 2.2.

Given $n - 1$ real-time channels which are schedulable over a link, to determine the minimum value of D_n for an n-th channel such that all the n channels are still schedulable over the link, we may first set $D_n = C_n$ and use Theorem 2.2 to check whether the channels are schedulable or not. If they are, C_n is the minimum delay bound for channel n. Otherwise, D_n must be increased according to the extent that the inequality of Theorem 2.2 is violated. This idea leads to the following theorem which solves Problem 2.2 under a preemptive deadline scheduling policy.

Theorem 2.3 (Solution to Problem 2.2) .
Let $f(t, D_n) = \sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i$ *and* S *be the set defined in Theorem 2.2 with* $D_n = C_n$. *Then,* $D_n = C_n$ *is the solution to Problem 2.2 if* $\forall t \in S$, $f(t, C_n) \leq t$. *Otherwise, the solution to Problem 2.2 is* $D_n = \max\{D^t : t \in G\}$, *where* $G = S \cap \{t : f(t, C_n) > t\}$ *and*

D^t is calculated as $D^t = C_n + k_f^t T_n + \epsilon_f^t + \epsilon_t^t$, with $k_f^t = \lceil (f(t, C_n) - t)/C_n \rceil - 1$. $\epsilon_f^t = f(t, C_n) - t - k_f^t C_n$, $\epsilon_t^t = t - C_n - k_t^t T_n$, $k_t^t = \lfloor (t - C_n)/T_n \rfloor$.

Proof: From Theorem 2.2, if $\forall t \in S$, $f(t, C_n) \leq t$, then all n channels are schedulable over the link with the n–th channel choosing $D_n = C_n$. Also, it is easy to see that D_n can not be smaller than C_n. Thus, $D_n = C_n$ is the solution to Problem 2.2.

Otherwise, we need to find a minimum D_n such that $\forall t \geq 0$, $f(t, D_n) \leq t$. Let $D_n = C_n + \delta D$ and $\delta f(t, \delta D) = \lceil (t - C_n - \delta D)/T_n \rceil^+ C_n - \lceil (t - C_n)/T_n \rceil^+ C_n$. Then, $f(t, D_n) = f(t, C_n) + \delta f(t, \delta D)$. For any $t_0 \in G$, define

$$f_{t_0}(t) = \begin{cases} 0 & \text{if } t < t_0 \\ f(t_0, C_n) & \text{if } t \geq t_0. \end{cases}$$

Then, $\forall t \geq 0$, $f(t, D_n) \leq t$ if and only if $\forall t \geq 0$, $\forall t_0 \in G$, $f_{t_0}(t) + \delta f(t, \delta D) \leq t$. So, we only need to find the minimum value of δD such that the latter inequality holds.

Let t_0 be a fixed point in G. We first find a minimum δD^{t_0} such that the inequality $f_{t_0}(t) + \delta f(t, \delta D^{t_0}) \leq t$ holds for all $t \geq 0$. Write $\delta D^{t_0} = k_d^{t_0} T_n + \epsilon_d^{t_0}$, $t_0 - C_n = k_t^{t_0} T_n + \epsilon_t^{t_0}$, with $k_d^{t_0} = \lfloor \delta D^{t_0}/T_n \rfloor$, $k_t^{t_0} = \lfloor (t_0 - C_n)/T_n \rfloor$. Since the first $n-1$ channels are schedulable over the link, we can restrict $\delta D^{t_0} \leq t_0 - C_n$. Under this condition, functions $f_{t_0}(t), \delta f(t, \delta D^{t_0})$ and t are plotted in Fig. 2.1. From Fig. 2.1, it is easy to see that $f_{t_0}(t) + \delta f(t, \delta D^{t_0}) \leq t$ holds for all $t \geq 0$ if and only if this inequality holds at $t = t_0$ and $t = t_1$, where $t_1 = t_0 + \epsilon_D^{t_0} - \epsilon_t^{t_0}$. Notice that $\delta f(t_0, \delta D^{t_0}) = -(k_d^{t_0} + 1)C_n$ and $\delta f(t_1, \delta D^{t_0}) = -k_d^{t_0} C_n$. Thus, $k_d^{t_0}$ and δD^{t_0} must satisfy the following two inequalities:

$$f(t_0, C_n) - (k_d^{t_0} + 1)C_n \leq t_0$$

$$f(t_0, C_n) - k_d^{t_0} C_n \leq t_1 = t_0 + \epsilon_d^{t_0} - \epsilon_t^{t_0}.$$

The values of $k_d^{t_0}$ and $\epsilon_d^{t_0}$ which satisfy the above inequalities and minimize $\delta D^{t_0} = k_d^{t_0} T_n + \epsilon_d^{t_0}$ are:

$$k_d^{t_0} = \lceil (f(t_0) - t_0)/C_n \rceil - 1 = k_f^{t_0}$$

$$\epsilon_d^{t_0} = f(t_0, C_n) - t_0 + \epsilon_t^{t_0} - k_d^{t_0} C_n = \epsilon_f^{t_0} + \epsilon_t^{t_0}.$$

The minimum δD^{t_0} thus obtained such that $f_{t_0}(t) + \delta f(t, \delta D^{t_0}) \leq t$, $\forall t \geq 0$ is $\delta D^{t_0} = k_d^{t_0} T_n + \epsilon_d^{t_0} = k_f^{t_0} T_n + \epsilon_f^{t_0} + \epsilon_t^{t_0}$.

Since $\delta f(t, \delta D)$ is a decreasing function of δD, the minimum value of δD that satisfies

$$f_{t_0}(t) + \delta f(t, \delta D) \leq t, \ \forall t \geq 0, \forall t_0 \in G$$

is thus

$$\delta D = \max\{\delta D^{t_0} : \ t_0 \in G\}.$$

This proves that $D = \max\{d^t : \ t \in G\}$ is the solution to Problem 2.2. □

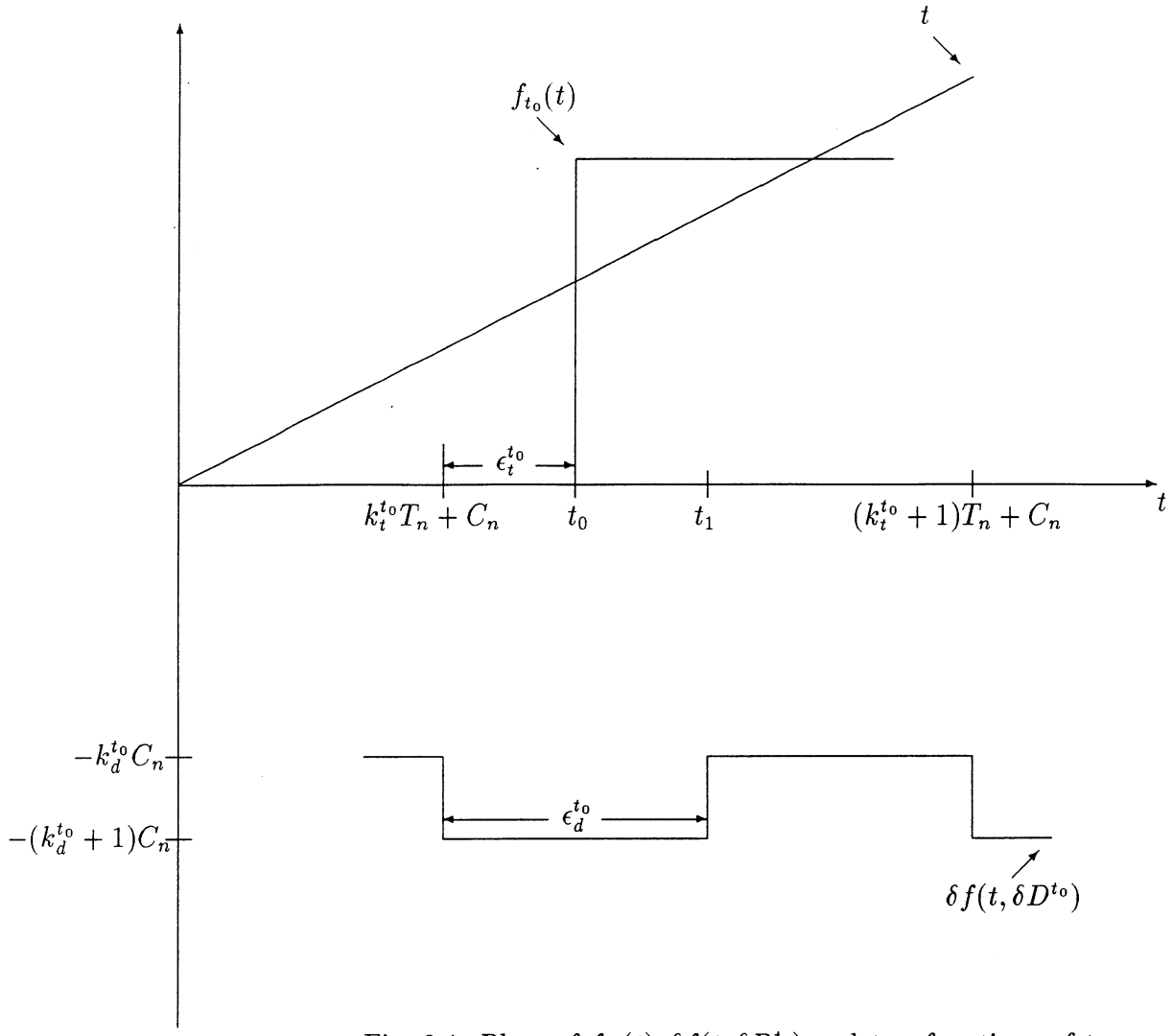

Fig. 2.1. Plots of $f_{t_0}(t), \delta f(t, \delta D^{t_0})$ and t as functions of t.

The following example shows the use of Theorem 2.3.

Example 2.2 .
A real-time channel $\tau_3 = (T_3, C_3, D_3) = (12, 3, D_3)$ *is to be established over a link over which two other channels* $\tau_1 = (T_1, C_1, D_1) = (10, 2, 5)$, $\tau_2 = (T_2, C_2, D_2) = (8, 4, 8)$ *have already been established. What is the minimum value of* D_3 *such that all three channels are schedulable over the link?*

Solution: The maximum utilization of the link $\sum_{i=1}^{3} C_i/T_i = 2/10 + 4/8 + 3/12 = 0.95 < 1$, so such a D_3 exists. Using Theorem 2.3, first let $D_3 = C_3 = 3$. Then, $f(t,3) = \lceil (t-5)/10 \rceil^+ 2 + \lceil (t-8)/8 \rceil^+ 4 + \lceil (t-3)/12 \rceil^+ 3$. From the definitions in Theorem 2.3, $t_{max} = 65$ and $S_1 = \{5, 15, 25, 35, 45, 55, 65\}$, $S_2 = \{8, 16, 24, 32, 40, 48, 56, 64\}$, $S_3 = \{3, 15, 27, 39, 51, 63\}$, $S = S_1 \cup S_2 \cup S_3$.

It is easy to verify that $f(t,3) \leq t$ holds over S except in $G = \{8, 16\}$ with $f(8,3) = 9$ and $f(16,3) = 18$. Using the formulae in Theorem 2.3, $D^8 = 9$ and $D^{16} = 6$. Thus, we get the solution $D_3 = \max\{D^t : t \in G\} = 9$. The correctness of the solution was

verified in Example 2.1. □

We give a comparison of Theorem 2.3 with that of [4, 6]. The result of [4] can not be used for this example since $C_1 + C_2 + C_3 = 9 > \min\{T_1, T_2, T_3\} = 8$. To compare the proposed scheme with that of [6], let us compute the worst–case delay for τ_3's messages from the priority–driven scheduling policy. Since τ_3 is not supposed to affect the delay guarantees for τ_1 and τ_2's messages, its messages must be assigned the lowest priority. Assigning τ_3's priority higher than τ_1 or τ_2 will cause one of τ_1 or τ_2's messages to miss its deadline when two messages, one for each channel, arrive simultaneously. Then, when messages of all three channels (one per channel) arrive at the link simultaneously, the delay of τ_3's messages will be 15. Thus, the solution given by [6] to Example 2 would be at least 15.[1] One can see that the gap between the sufficient condition of [6] and our sufficient and necessary condition for the schedulability of real–time channels is quite large, indicating the superiority of the latter.

The complexity of the solutions to Problems 2.1 and 2.2 presented in Theorems 2.2 and 2.3 is of the order of the size of set S which, from its definition in Theorem 2.2, is reasonably small when the maximum link utilization $\sum_{i=1}^{n} C_i/T_i$ is not too close to one. However, as the link utilization $\sum_{i=1}^{n} C_i/T_i$ approaches one, S could become very large. One way to deal with this problem is to avoid the heavily–loaded links in the network (e.g., with the utilization > 0.9) during the real–time channel routing phase. This is based on the reasoning that links with high utilizations induce large delays, and thus, new real–time channels should not be routed through them. We also do not wish the link utilization to reach one because this could block non real–time messages completely from these links.

If the size of S is still thought to be too large even for moderately loaded links, one can use the following sufficient condition for the schedulability of n channels over a link that only needs to validate an inequality on at most n points.

A set of channels $\tau_i = (T_i, C_i, D_i), i = 1, 2, ..., n$, is said to be *ordered* if $D_1 \leq D_2 \leq \cdots \leq D_n$.

Theorem 2.4 (Sufficient condition for Problem 2.1) .
A set of ordered channels $\tau_i = (T_i, C_i, D_i)$, $i = 1, ..., n$, are schedulable over a link with the preemptive deadline scheduling algorithm if $\sum_{i=1}^{n} C_i/T_i < 1$, and

$$\forall k \in K, \quad \sum_{i=1}^{k} (1 + (D_k - D_i)/T_i) C_i \leq D_k,$$

where $K = \{1, 2, ..., n\} - \{k : D_k = D_{k+1}, 1 \leq k \leq n-1\}$.

Proof: We want to prove that satisfaction of the above inequality implies that of the condition of Theorem 2.1. Let $K = \{k_1, ..., k_m\}$ such that $k_1 < k_2 < \cdots < k_m$, and let $D_{k_0} = 0$, $D_{k_{m+1}} = \infty$. Then, $\forall\, t \in [D_{k_j},\ D_{k_{j+1}})$,

$$\begin{aligned} \sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i - t &= \sum_{i=1}^{k_j} \lceil (t - D_i)/T_i \rceil^+ C_i - t \\ &\leq \sum_{i=1}^{k_j} (1 + (t - D_i)/T_i) C_i - t \end{aligned}$$

[1] Actually, since 15 is larger than T_3, [6] cannot give a solution for Example 2.

$$
\begin{aligned}
&= (\sum_{i=1}^{k_j} C_i/T_i - 1)t + \sum_{i=1}^{k_j}(1 - D_i/T_i)C_i \\
&\leq (\sum_{i=1}^{k_j} C_i/T_i - 1)D_{k_j} + \sum_{i=1}^{k_j}(1 - D_i/T_i)C_i \\
&= \sum_{i=1}^{k_j}(1 + (D_{k_j} - D_i)/T_i)C_i - D_{k_j} \\
&\leq 0.
\end{aligned}
$$

Since the above inequality holds for $j = 0, 1, ..., m$, the theorem follows. □

An interesting by-product of Theorem 2.4 is a sufficient condition for the channel schedulability when D_i/T_i equals a constant θ, called the the *system hazard*, for all $1 \leq i \leq n$. Peng [16] proved that a sufficient schedulability condition is $\theta \geq \sum_{i=1}^{n} C_i/T_i$. Using Theorem 2.4, we have another sufficient condition in terms of the system hazard:

$$\theta \geq \max_{k \in K}\{(1 + (1 - \sum_{i=1}^{k} C_i/T_i)/\sum_{i=1}^{k} C_i/T_k)^{-1}\}.$$

Since $C_i/T_k \leq C_i/T_i$, it can be proved that the above sufficient condition is tighter than Peng's condition in [16]. The following example shows how much of improvement can be achieved. Suppose there are two channels,

- $\tau_1 : T_1 = 4,\ C_1 = 2,$
- $\tau_2 : T_2 = 16,\ C_2 = 4.$

Using Peng's condition, the minimum system hazard θ is $\sum_{i=1}^{2} C_i/T_i = 0.75$. From our condition, θ can be as small as 0.6. This shows the tightness (superiority) of the sufficient condition of Theorem 2.4.

It is worth pointing out that the simplicity of Theorem 2.4 is achieved at the cost that it gives only sufficient, but not necessary, conditions. Violation of sufficient conditions does not necessarily mean that the channels are not schedulable over the link. Thus, Theorem 2.4 should be used only when the simplicity of the algorithm is more important than the tightness of the results.

A set of channels is said to be *strongly schedulable* over a link if the conditions of Theorem 2.4 are satisfied. Then, we have the following problem which is a modified version of Problem 2.2.

Problem 2.3 (Strong schedulability) .
Suppose $n-1$ channels, $\tau_i = (T_i, C_i, D_i)$, $i = 1, 2, ..., n-1$, are strongly schedulable over a link. Given a new channel τ_n with the minimum message inter-arrival time T_n and maximum message transmission time C_n, what is the minimum value of D_n such that all n channels are still strongly schedulable?

The following theorem solves Problem 2.3.

Theorem 2.5 (Solution to Problem 2.3) .
Let $D_n = C_n$. If for $k = 1, \cdots, n$, $\delta_k = \sum_{D_i \leq D_k}(1 + (D_k - D_i)/T_i)C_i - D_k \leq 0$, then $D_n = C_n$ is the solution to Problem 2.3. Otherwise, let $K_G = \{k : \delta_k > 0\}$. The solution to Problem 2.3 is $D_n = \max\{D_n^k : k \in K_G\}$, where $D_n^k = C_n + (T_n/C_n)\delta_k$ if $C_n + (T_n/C_n)\delta_k < D_k$, otherwise, $D_n^k = D_k + (C_n - D_k + (T_n/C_n)\delta_k)/(1 + (1 - \sum_{D_i \leq D_k} C_i/T_i)(T_n/C_n))$.

We are not going to give the proof of Theorem 2.5 since it follows the same idea as that of Theorem 2.3.

For the purpose of comparison, we re-do Example 2.2 with Theorem 2.5. First, set $D_3 = C_3 = 3$. Here only two points need to be checked: $\delta_1 = (1 + (D_1 - D_3)/T_3)C_3 + C_1 - D_1 = 0.5 > 0$, and $\delta_2 = (1+(D_2-D_1)/T_1)C_1+(1+(D_2-D_3)/T_3)C_3+C_2-D_2 = 57/20 > 0$. Thus, $K_G = \{1, 2\}$. From Theorem 2.5, $D_3^1 = 5$ and $D_3^2 = 13.3$. So $D_3 = 14$ is the minimum integer–valued delay bound which can be assigned to channel τ_3 such that the three channels are strongly schedulable over the link.

This example shows that the result obtained from Theorem 2.5 is usually not as good as that obtained from Theorem 2.3. However, since the computational complexity of Theorem 2.5 is in the order of the number of channels to be scheduled, Theorem 2.5 is useful when a smaller channel establishment time is required.

In this example, the solution obtained from Theorem 2.5 is shown to be better than that of [6] (see Example 2.2). Although this cannot be said in general, our solution always requires less computation than that of [6].

2.4 Non-preemptive Deadline Scheduling

All results obtained in the last section are based on an assumption that a preemptive deadline scheduling policy is used for message transmissions. This means that the transmission of a message may be interrupted at any time when another message with an earlier deadline is generated, and resumed later from where it was interrupted. Unlike the task scheduling on a processor, this is difficult to implement in practice. A more realistic method is to break a message into packets and the preemption of the message transmission is allowed only at the boundaries of the packets. In other words, each packet is assigned a deadline and a *non-preemptive* deadline scheduling policy is used for the transmission of the packets. In this section, we will derive solutions to the schedulability and minimum delay bound problems under this policy.

We assume that the packets of a message are generated contiguously. That is, the next packet of a message is always ready for transmission when the first packet of the message is transmitted. The generation time of a message is defined as the time when its first packet is ready for transmission. The deadlines of the packets of a message belong to a real–time channel $\tau = (T, C, D)$ is calculated as the message generation time plus D. Let C_p denote the time needed to transmit a maximum–size packet. Then, the following theorem gives a solution to the schedulability problem under the non-preemptive deadline scheduling policy.

Theorem 2.6 (Solution to the schedulability problem) .
In the presence of non real–time packets, a set of real–time channels $\tau_i = (T_i, C_i, D_i)$, $i = 1, 2, \cdots, n$, are schedulable over a link under the non-preemptive deadline scheduling policy if and only if

$$\forall\, t \geq D_{min},\ \sum_{j=1}^{n} \lceil (t - D_j)/T_j \rceil^+ C_j + C_p \leq t,$$

where $D_{min} = \min\{D_i : 1 \leq i \leq n\}$.

Proof of the necessary condition: We prove this by contradiction. Suppose the channels are schedulable over the link and there exists a $t_0 \geq D_{min}$ such that $\sum_{j=1}^{n} \lceil (t_0 -$

$D_j)/T_j\rceil^+C_j + C_p > t_0$. Consider the following scenario. The link has been idle for all $t < 0$. At $t = 0^-$, a maximum–size non real–time packet arrived at the link and then began transmitting at $t = 0$ (would last for C_p seconds). Starting from $t = 0$, all the real–time channels' messages are generated at their maximum rates. Then, the time needed to transmit all these real–time packets with deadlines $\leq t_0$ is $\sum_{j=1}^{n}\lceil(t_0 - D_j)/T_j\rceil^+C_j$. Since all n channels are schedulable over the link, these packets must be transmitted before time t_0, i.e.,

$$\sum_{j=1}^{n}\lceil(t_0 - D_j)/T_j\rceil^+C_j + C_p \leq t_0.$$

This contradicts the assumption.

Proof of the sufficient condition: We use contradiction again. Suppose the inequality of the theorem holds, but the channels are not schedulable over the link. Then, there exists a t_1 such that a real–time packet is being transmitted at time $t = t_1$ and misses its deadline. Let $t_0 \leq t_1$ be the *earliest* time such that in the time interval $[t_0, t_1]$, the link is busy transmitting only those packets with deadlines $\leq t_1$. Thus, at $t = t_0^-$, the link is either idle, transmitting a non real–time packet, or transmitting a real–time packet with deadline $\geq t_1$.

If the link is idle at $t = t_0^-$, then $t_1 - t_0 \geq D_{min}$ since a message generated after t_0 misses its deadline at t_1. Also, during the time interval $[t_0, t_1]$, the link is busy transmitting only those messages which are generated at the link during the time interval $[t_0, t_1]$ and with deadlines $\leq t_1$. The maximum time needed to transmit these messages is $T = \sum_{j=1}^{n}\lceil(t_1 - t_0 - D_j)/T_j\rceil^+C_j$. Since one message misses its deadline at t_1, this T must be larger than $t_1 - t_0$, that is,

$$\sum_{j=1}^{n}\lceil(t_1 - t_0 - D_j)/T_j\rceil^+C_j > t_1 - t_0.$$

By letting $t = t_1 - t_0$, one can see that the above inequality contradicts the inequality of the theorem.

If the link is transmitting a non real–time message at $t = t_0^-$, then $t_1 - t_0 + C_p \geq D_{min}$ since a message generated after $t_0 - C_p$ misses its deadline at t_1. Also, during the time interval $[t_0, t_1]$, the link is busy transmitting only those messages generated at the link during the time period $[t_0 - C_p, t_1]$ (this corresponds to the worst-case that the non real–time packet has the maximum length) and having deadlines $\leq t_1$. The maximum time needed to transmit these messages is $T = \sum_{j=1}^{n}\lceil(t_1 - t_0 + C_p - D_j)/T_j\rceil^+C_j$. Since there is a message missing its deadline at t_1, this T must be larger than $t_1 - t_0$, i.e.,

$$\sum_{j=1}^{n}\lceil(t_1 - t_0 + C_p - D_j)/T_j\rceil^+C_j > t_1 - t_0.$$

By letting $t = t_1 - t_0 + C_p$, one can see that the above inequality contradicts the inequality of the theorem with $i = 0$.

Now suppose the link is transmitting a message belonging to a real–time channel i_0 at $t = t_0^-$. From the definition of t_0, this message must have been generated at the link at time $t_0^- - C_p$ (again, this corresponds to the worst-case that the real–time packet is of the maximum length) and has a deadline $> t_1$. Then, $d_{min} \leq t_1 - t_0 + C_p < D_{i_0}$. Also, during the time interval $[t_0, t_1]$, the link is busy transmitting only those messages generated

at the link during the time period $[t_0 - C_p, t_1]$ and have deadlines $\leq t_1$. The maximum time needed to transmit these messages is $T = \sum_{j=1}^{n} \lceil (t_1 - t_0 + C_p - D_j)/T_j \rceil^+$. Since there is a message missing its deadline at t_1, this T must be larger than $t_1 - t_0$, i.e.,

$$\sum_{j=1}^{n} \lceil (t_1 - t_0 + C_p - D_j)/T_j \rceil^+ > t_1 - t_0.$$

By letting $t = t_1 - t_0 + C_p$, one can see the above inequality contradicting the inequality of the theorem. □

Results similar to Theorems 2.2 – 2.5 for the non-preemptive scheduling policy can be obtained from Theorem 2.6 as follows.

Theorem 2.7 (Practical solution to the schedulability problem) .
In the presence of non real-time packets, a set of real-time channels $\tau_i = (T_i, C_i, D_i)$, $i = 1, 2, ..., n$, *are schedulable over a link under the non-preemptive deadline-driven scheduling policy if and only if both of the following hold:*

1. $\sum_{j=1}^{n} C_j/T_j \leq 1$.
2. $\forall t \in S$, $\sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^+ C_i + C_p \leq t$,
   where $S = \cup_{i=1}^{n} S_i$, $S_i = \{D_i + nT_i : \ n = 0, 1, ..., \lfloor (t_{max} - D_i)/T_i \rfloor\}$,
   and $t_{max} = \max\{D_1, ..., D_n, (C_p + \sum_{i=1}^{n}(1 - D_i/T_i)C_i)/(1 - \sum_{i=1}^{n} C_i/T_i)\}$.

Theorem 2.7 allows us to check only a finite number of points to verify the schedulability of real-time channels. The following theorem gives the solution to the minimum delay bound problem (Problem 2.2) under the non-preemptive scheduling policy.

Theorem 2.8 (Solution to the minimum delay bound problem) .
Let $f(t, D_n) = \sum_{j=1}^{n} \lceil (t - D_j)/T_j \rceil^+ C_j + C_p$ *and* S *be the set defined above with* $D_n = C_n + C_p$. *Then,* $D_n = C_n + C_p$ *is the worst-case delay if* $f(t, C_n) \leq t$, $\forall t \in S$. *Otherwise, the worst-case delay is* $D_n = \max\{D^t : t \in G\}$, *where* $G = S \cap \{t : f(t, C_n) > t\}$ *and* D^t *is computed as* $D^t = C_n + k_f^t T_n + \epsilon_f^t + \epsilon_t^t$, *with* $k_f^t = \lfloor (f(t, C_n) - t)/C_n \rfloor$, $\epsilon_f^t = f(t, C_n) - t - k_f^t C_n$, $\epsilon_t^t = t - C_n - k_t^t T_n$, $k_t^t = \lfloor (t - C_n)/T_n \rfloor$.

Also, we have the sufficient conditions for the schedulability of real-time channels under a non-preemptive scheduling policy.

Theorem 2.9 (Sufficient conditions for the schedulability problem) .
A set of ordered channels $\tau_i = (T_i, C_i, D_i)$, $i = 1, ..., n$, *are schedulable over a link with the non-preemptive deadline-driven algorithm if* $\sum_{i=1}^{n} C_i/T_i < 1$ *and*

$$\sum_{i=1}^{k} (1 + (D_k - D_j)/T_j) C_j + C_p \leq D_k, \quad \forall k \in K,$$

where $K = \{1, 2, \cdots, n\} - \{k : D_k = D_{k+1}, 1 \leq k \leq n - 1\}$.

Similarly, a set of channels is said to be *strongly schedulable* under a non-preemptive deadline scheduling policy if the conditions of Theorem 2.9 are satisfied. The following theorem solves Problem 2.3.

Theorem 2.10 (Solution to the strong schedulability problem) .
Let $D_n = C_n + C_p$. *If for* $i = 0, \cdots, n$, $k = 1, \cdots, n$, $\delta_k(i) = \sum_{D_j \leq \min\{D_i, D_k\}} (1 + (D_k - D_j)/T_j)C_j + C_i - D_k \leq 0$, *then* $D_n = C_n + C_p$ *is the solution to Problem 2.3. Otherwise, let* $K_G^i = \{k : \delta_k(i) > 0\}$. *The solution to Problem 2.3 is* $D_n = \max_{0 \leq i \leq n} \{\max\{D_n^k(i) : k \in K_G^i\}$, *where* $D_n^k(i) = C_n + (T_n/C_n)\delta_k(i)$ *if* $C_n + (T_n/C_n)\delta_k(i) < D_k$, *otherwise,* $D_n^k = D_k + (C_n - D_k + (T_n/C_n)\delta_k(i))/(1 + (1 - \sum_{D_i \leq D_k} C_i/T_i)(T_n/C_n))$.

We do not present the proofs of Theorems 2.7 – 2.9 here since using Theorem 2.6, they are basically the same as the proofs of Theorems 2.2 – 2.5.

2.5 Conclusion

Solutions to the channel schedulability and minimum delay bound problems are derived in this chapter, which form the mathematical basis for the real–time channel concept. The real–time channel protocols are going to be constructed on these results in the next chapter.

Application of the deadline scheduling theory is not limited to real–time channels only. A link can represent any time–critical resource like a processor, and the channels can represent any periodic time–critical tasks. So, our solutions can also be used for periodic task scheduling problems as those discussed in [5]. Actually, the problems addressed in this chapter are more general than the ones in [5] since we do not require the delay bound D to be equal to the task period T and the tasks do not have to be strictly periodic as long as there exists a minimum task inter-arrival time.

CHAPTER 3

REAL–TIME CHANNEL APPROACH

3.1 Introduction

As discussed in Chapter 1, real–time channels use two techniques to guarantee the end–to–end message delay bound: admission control via the channel establishment procedure and the deadline scheduling of message transmissions. Thus real–time channels are realized with two protocols: a channel establishment protocol and a message transmission protocol.

The channel establishment protocol handles requests for the establishment of real–time channels. It checks whether the requested end–to–end message delivery delay bound can be guaranteed for a real–time channel under the current network–load condition. A channel establishment request is granted only if the requested delay bound can be guaranteed. The message transmission protocol implements the deadline scheduling of message transmissions. It specifies how a message is divided into packets, and how deadlines of a packet over the links it traverses are calculated. These two parts are closely related. The calculation of the end–to–end message delivery delay bound depends on the transmission protocol used, and the transmission protocol must be designed such that the requested delay bound can be guaranteed and easily verified.

In Section 3.2, we will present in detail the message transmission protocol with which the message delivery delay bound over each link can be guaranteed and verified. Then, both a centralized and a distributed version of the real–time channel establishment protocol are constructed in Section 3.3. The centralized protocol requires a special node, called the Network Manager (NM), to take care of the channel establishment requests, while with the distributed protocol, the channel establishment algorithm is executed in a decentralized manner. The chapter concludes with Section 3.4.

3.2 Message Transmission Protocol

Due to its ease of implementation, a non-preemptive deadline scheduling policy is used for the message transmission protocol. Specifically, each message that belongs to a real–time channel is divided into packets and the packets are transmitted individually. The message transmission protocol assigns each packet a deadline over each link it traverses. When several packets contend for the same transmission link, the one with the earliest deadline is transmitted first. The in-progress transmission of a packet will not be interrupted.

One key problem of the message transmission protocol is the calculation of packet link deadlines. As discussed in Chapter 1, the advantages of using the deadline scheduling

policy are the minimal effects of contention delays and the protection among the established channels. However, these advantages cannot be realized unless the link deadlines are assigned such that the following conditions are satisfied.

C1. Meeting the link deadlines guarantees the end–to–end message delivery delay bound.

C2. The deadline scheduling theory of Chapter 2 can be used to determine whether or not each packet can meet its link deadlines.

C3. Violation of the message generation pattern of one real–time channel will not affect the guarantees to other channels.

C1 is the correctness condition. The link deadlines would be meaningless if they cannot guarantee the end–to–end message delay bound. C2 is the solvability condition which can check the schedulability of real–time channels. C3 ensures the channel protection as discussed in Chapter 1, which is important since real–time channels would be of little use in practice if one malicious source can disrupt the regular operations of other channels.

In Chapter 2, the deadlines of the packets of a message are calculated as the message generation time plus the requested end–to–end message delivery delay bound. This scheme applies to single–link real–time channels only and does not satisfy C3. For multi-link real–time channels, an easy way to meet C1 would be to divide the requested end–to–end message delay bound into link delay bounds, and assign the deadline of a packet over a link to be the arrival time of the packet at the link plus the link delay bound. Clearly, in this way meeting the link deadlines guarantees the end–to–end delay bound. Unfortunately, this simple scheme does not satisfy C2 and C3.

Recall that in verifying whether or not the packet deadlines will be met using the deadline scheduling theory of Chapter 2, the following assumptions are used: (A1) the packets of a message are generated at the same time, and (A2) there exist a minimum message inter-arrival time and a maximum message size for each real–time channel. However, these assumptions are not always true because: (1) when a message is split into packets which are transmitted individually, the packets may not always arrive at an intermediate link as a continuous stream, (2) a real–time channel may generate messages faster and larger than the specified values T and M, and (3) even if no real–time channels violate their traffic specifications, the message inter-arrival times at an intermediate link could be smaller than T due to the possible uneven queueing delays at upstream links. Thus, with the simple packet deadline assignment scheme described above, the deadline scheduling theory of Chapter 2 cannot be directly used to check if the packet deadlines will be met (violating C2), and the real–time channels are not protected from one another (violating C3).

By convention, we say a message is *generated* at the source node and a message *arrives* at an intermediate node or a destination node. To solve the problem described above, we use the *logical* message generation/arrival times to calculate the packet deadlines. At a source node, if a message is generated too early (i.e., with the message inter-generation time smaller than T), its logical generation time is set to be later than its actual generation time as if it were generated on time. If a message is generated with too large a size (i.e., larger than M bits), the extra part of the message is considered to be part of the next message. The deadlines of a message's packets are then assigned to be the logical message generation time plus the link delay bound. Notice that with the deadline scheduling policy, packets of different channels interfere with each other only through their deadlines. Thus, with the packet deadlines calculated from their logical generation times, a real–time channel may

never seem to violate its pre-specified traffic generation pattern from to other channels' points of view.

At an intermediate node, the logical arrival time of a message is calculated as the time the message would have arrived at the node if each of its packets had experienced the maximum delays over up-stream links. In this way, the adverse effect of the early-arrived messages is removed. To the packets of the other channels, the message inter-arrival time of a channel will never appear to be smaller than T. Also, all packets of a message are always available for transmission after the logical arrival time of the message. Thus, the deadline scheduling theory of Chapter 2 can be used to check the schedulability conditions at intermediate links.

Calculation of the packet deadlines from the logical generation/arrival times will not introduce any *unnecessary* packet delays. An early generated/arrived packet is eligible for transmission before its logical generation/arrival time if there are no other packets with earlier deadlines contending for use of the transmission link. An early-arrived packet will be delayed for transmission only when its early transmission could cause adverse effect to other channels.

Suppose a real-time channel is to be established over n links $\ell_1, \cdots, \ell_n$. The requested end-to-end delay bound D is split into link delay bounds. The algorithm for doing this will be given in the next section. Suppose the link delay bound over ℓ_i is d_i, $i = 1, \cdots, n$. We now present the message transmission protocol.

Source node message transmission protocol: In addition to the conventional packet header information, a packet also contains the following header fields exclusively for real-time channels: (1) logical packet arrival/generation time t_l, (2) packet deadline t_d, (3) routing information which contains the channel's transmission link IDs ℓ_i, and (4) scheduling information which contains the link delay bounds d_i.

The source node uses variables t_p, t_c to record the times when the previous and current messages of the channel are generated, and m_r, m_a to record the size (in bits) of the remaining message (i.e., the part not yet packetized) and the accumulated message size (to detect an oversize message), respectively. Suppose the parameters of the real-time channel are (T, M, D), and P is the packet size with which the messages will be divided. Initialize $t_p := -\infty$, $m_r := 0$, $m_a := 0$.

The message transmission protocol at the source node is given as follows:

Protocol 3.1 (Message transmission at source node) .

Step 1: *When a message of m bits is generated at time t, set the current message generation time $t_c := t$, and the remaining message size $m_r := m$.*

Step 2: *Construct a packet of size $p = \min\{P, m_r\}$. Fill the packet routing and scheduling information ℓ_i's and d_i's in the packet header fields. Update the accumulated message size $m_a := m_a + p$. Set the logical generation time $t_l := \max\{t_c, t_p + T\} + \lfloor m_a/(M+1) \rfloor T$ and the deadline $t_d := t_l + d_1$.*

Step 3: *Forward the constructed packet to the transmission link identified by the link ID ℓ_1. Update the remaining message size $m_r := m_r - p$. If $m_r > 0$, goto Step 2 to assemble the next packet. Otherwise, update $m_a := \max\{m_a - M, 0\}$, $t_p := \max\{t_c, t_p + T\}$. and wait for the generation of the next message.*

The function performed at the source node is to police the message generation of a real–time channel. If the messages are generated too fast or with too large sizes, the extra parts of the messages will be assigned later deadlines as if they were generated regularly (see Step 2). As discussed before, with the deadline scheduling policy, the packets of different channels interfere with each other only through their deadlines. With the packet deadlines calculated from Protocol 3.1, the violation of the traffic generation pattern of one channel will not affect other channels and the real–time channels are protected from each other. Notice that this kind of channel protection is not achievable with other scheduling policies like the FIFO, priority scheduling, or exhaustive round–robin. In this sense, the importance of using the deadline scheduling policy cannot be over-emphasized and the deadline scheduling of message transmission is a key to of real–time channels.

The packets forwarded to an outgoing link are scheduled according to their deadlines. We will discuss in Chapter 4 this scheduling process and present an implementation of the deadline scheduler. The transmitter will put a time-stamp t_t (the time when the first bit of the packet is transmitted) at the header of each packet which will be used by the next node to remove any clock skew between two nodes.

Intermediate node message transmission protocol: With logical message generation times as calculated from Protocol 3.1 at the source node, it is guaranteed that the message inter-generation time of a real–time channel (T, M, D) will never be smaller than T. But, if two messages experience different delays over up-stream links, they could still arrive at an intermediate link with an inter-arrival time smaller than T. So, logical message arrival times at intermediate links must be assigned such that the logical message inter-arrival time (the time between the logical arrival times of two consequent messages) is always not smaller than T. As discussed before, this can be done by setting the logical arrival time of a message to be the time the message would have arrived if each of its packets had experienced the maximum delays over the up-stream links. In this way, the the time between the logical arrival times of two packets is always the same as the time between the logical generation times of the packets. Thus the problem of uneven packet delays at up-stream links can be overcome.

The message transmission protocol at an intermediate node, say node i, is presented below. Let t denote the local time at node i when the first bit of a packet arrives. An arriving packet is processed as follows.

Protocol 3.2 (Message transmission at intermediate node) .

Step 1: *Calculate the clock skew between node $i-1$ and node i: $t_s = t - t_t$.*[1] *Suppose link ℓ_{i-1} has a transmission bandwidth R_{i-1}, then update the packet's logical arrival time $t_l := t_l + t_s + d_{i-1} - \max\{0, (M-P)/R_{i-1}\}$, and deadline $t_d = t_l + d_i$.*

Step 2: *Remove ℓ_{i-1} and d_{i-1} from the packet header and forward the packet to the link ℓ_i.*

The calculation of the logical arrival time t_l in Step 1 is explained in Fig. 3.1 which shows the logical arrival times of a message at node $i-1$ and i. A message's *link delay* is defined as the time between the logical arrival time of the message and the time when the last bit of the message is transmitted. Then, a message's logical arrival time at

[1] As will be discussed in Section 3.3, this also removes the propagation delay between two nodes.

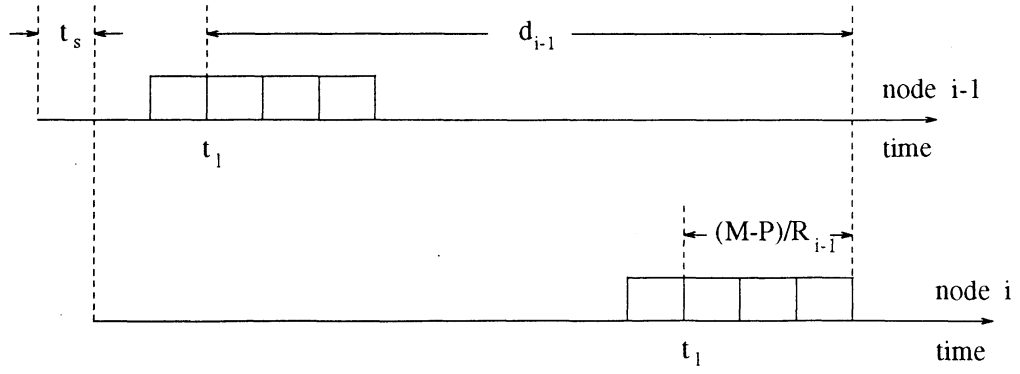

Figure 3.1: Calculation of a message's logical arrival time at node i.

node i equals the message's logical arrival time at node $i-1$ plus the worst-case link delay d_{i-1} minus the time needed to transmit $M-P$ bits (i.e., the maximum message minus the first packet). One important fact is that if all channels are correctly established, which ensures the messages' actual link delays not to exceed the worst-case link delays d_i's, then the actual message arrival times at a node will never be later than the corresponding logical arrival times. If this is violated, then something must have gone wrong: either a real–time channel is not correctly established or a source node is sending real-time messages without establishing a real–time channel first.

3.3 Channel Establishment Protocol

The concept of logical message generation/arrival times introduced in Section 3.2 enables us to check the queueing delay bound over a single link using the deadline scheduling theory of Chapter 2. In this section, we show how the end–to–end message delivery delay, which is defined as the time between the generation of a message at the source node and the reception of the last bit of the message at the destination node, can be guaranteed to be below a requested bound using the the real–time channel establishment protocol.

The end–to–end packet delivery delay is composed of the following components:

- Switching delay: the time needed to move a packet from an incoming link to an appropriate outgoing link.
- Scheduling delay: the time needed to insert the arriving packet in a priority queue at the transmission link, according to a specific scheduling policy.
- Queueing delay: the time between the insertion of the packet at the priority queue (i.e., when the packet is ready for transmission) and the transmission of the last bit of the packet.
- Propagation delay: the time needed for the packet to reach the next node.

A delay component is said to be *inherent* if it cannot be controlled under a certain bound without a proper high level traffic control scheme. In other words, the inherent part of the delay cannot be removed by using an ideal network which has transmission links and switching nodes of arbitrarily high speed. In the development of the real–time channel protocol, the inherent part of a packet's end–to–end delay is found to be the queueing delay.

This is because the the incoming and outgoing links always have the same transmission rate in a general network. So, no matter how fast the link transmission speed may be, queueing delays at an transmission link cannot be bounded when several incoming links send packets at their fastest speed to an outgoing link continuously. In contrast, the switching and scheduling delays of a packet can always be controlled under a certain bound by increasing the switching/processing power of the node. For example, if the switching/processing speed of a node with n incoming links is n times as fast as the transmission speed of its links, the switching and scheduling delays of a packet will never exceed the packet transmission time.

Notice some overlap between the switching/scheduling delay and the queueing delay since the switching/scheduling of a packet can start at the reception of the packet header. Thus if the switching/scheduling can be done before the arrival of the last bit of the packet, the switching/scheduling time can be completely ignored. We will use this assumption in the rest of this chapter. The design of a fast switching that satisfies this assumption will be discussed in the next chapter.

The propagation delay, denoted by D_p, is a constant over a given route and can usually be ignored if the physical distance between the source and destination is not too long. If D_p is not negligible compared to the requested end–to–end delay bound D, it can be pre-subtracted from D and establish a real–time channel with a new end–to–end delay bound $D - D_p$. From Protocol 3.2, the propagation delay between two neighboring nodes will be automatically removed as the clock skew, thus guaranteeing the actual delay bound D.

From the above discussions, we assume that the end–to–end delay is composed of the queueing delays at the transmission links only. To establish a real–time channel over a given route, the first step is to divide the end–to–end delay bound into link delay bounds. The following theorem gives a relation between the link delay bounds and end–to–end delay bound.

Theorem 3.1 (End–to–end Delay Bound) .
Suppose a real–time channel with a maximum message transmission time C runs over n links which guarantee the worst–case message link delays $d_1, \cdots, d_n$, respectively. Then, the end–to–end message delivery delay is bounded by

$$D = \sum_{i=1}^{n} d_i - (n-1)\max\{0, (C - C_p)\},$$

where C_p is the packet transmission time.

Proof: Since the end–to–end delay of a message will not exceed the end–to–end delay of a maximum–size message which experiences the worst–case link delays on the route, we only need to show that the latter will not exceed D as calculated in the theorem.

Label the nodes such that link i runs from node $i-1$ to node i. Let t_i denote the time the last bit of the message arrives at node i, and let τ_i denote the time the last bit of the message's first packet arrives at node i. Then, from the definition of the end–to–end message delay, we get $D = t_n - t_0$, and from the definition of the link delay, we have $d_i = t_i - \tau_{i-1}$. At the source node (i.e., node 0) we assume that all packets of the message are generated at the same time. Thus, $t_0 = \tau_i$. For $i > 0$, $t_i \geq \tau_i + \max\{0, (C - C_p)\}$ because Packets arrive sequentially. Thus, we have $D = t_n - t_0 = \sum_{i=1}^{n}(t_i - t_{i-1}) = \sum_{i=1}^{n}(t_i - \tau_{i-1} + \sum_{i=1}^{n}(\tau_{i-1} - t_{i-1}) \leq \sum_{i=1}^{n} d_i - (n-1)\max\{0, (C - C_p)\}$. The equality holds

when all the packets arrive in a head-to-tail fashion. □

Theorem 3.1 shows that the end–to–end delay bound D is a decreasing function of the packet size. Thus the end–to–end delay can be reduced by dividing a message into small packets. This is one of the reasons why the packet–switched transmission is almost exclusively used in computer networks. However, one should not draw a conclusion from Theorem 3.1 that the smaller the packet size the better. Too small packets cause two problems. First, since each packet needs a header whose size is independent of the packet size, the smaller the packet, the more overhead is introduced, and the lower transmission efficiency. Secondly, small packet size increases the burden of switching nodes. As discussed before, the real–time channels require that each packet be switched/scheduled in a time period no longer than the packet transmission time. Smaller packets need faster switching nodes. Thus, unless a very tight end–to–end delay bound is required or the number of links of a channel is large, the packet size should not be set too small.

Using Theorem 3.1 and the deadline scheduling theory given in Chapter 2, there are two ways to guarantee the requested end–to–end message delay bound of a real–time channel.

M1: Divide the end–to–end delay bound D into link delay bounds d_i's such that the equation of Theorem 3.1 is satisfied. Check the schedulability of the channels over each link using Theorem 2.7. If all the checks are positive, the requested real–time channel can be established. Otherwise, the channel request is rejected.

M2: Using Theorem 2.8, calculate the minimum link delay bound d_i over each link of the channel. If the end–to–end delay bound calculated from Theorem 3.1 using these minimum link delay bounds is not larger than D, the requested channel can be established. Otherwise, the channel request is rejected.

M2 is in general superior to M1 since it calculates the minimum end–to–end delay bound which can be guaranteed under the current network–load condition. Thus, if a real–time channel cannot be established using M2, it cannot be established using M1 either under any link delay assignment policy. However, M1 is useful in situations where the network is lightly loaded (thus the requested channel is always establishable) and the client has specific preference on the link delay assignment policy. Also, M1 is simpler than M2 and is more suitable for the distributed real–time channel establishment protocol.

We now formally present the real–time channel establishment protocol. The centralized protocol is given first.

Protocol 3.3 (Centralized channel establishment) .

Step 1: To establish a real–time channel, the source node sends a channel–request message that contains the channel parameters (T, M, D) and the addresses of the source and destination nodes to a special node containing the *Network Manager* (NM), which manages all the real–time channels in the network.

Step 2: The NM maintains a channel table containing the information about all established real–time channels. After receiving a channel–request message,

Step 2.1: The NM selects a route from the source to the destination over which the real–time channel is to be established. Suppose the selected route contains k links $\ell_1, \cdots, \ell_k$.

Step 2.2: For $i = 1, \cdots, k$, the NM calculates the message link delay bound d^i_{max} over link ℓ_i using Theorem 2.8. Then, it calculates the end-to-end delay bound D_{max} using Theorem 3.1.

Step 2.3: If $D_{max} \leq D$, the NM sets the delay bound over ℓ_i to be $d_i = d^i_{max} + (D - D_{max})/k$.

Step 3: If $D_{max} > D$, the NM sends the source node a channel-reject message notifying the denial of the channel request. Otherwise, the NM sends the source node a channel-accept message which contains the route information and the assigned link delay bounds d_i's. After receiving the channel-accept message, the source node can start message transmission using Protocol 3.1.

M1 is used in the above protocol due to its "optimality" in checking the channel establishment conditions. The reason for re-assigning the link delay bounds in Step 2.3 is to make the link delay bounds as large as possible to facilitate the establishment of future channels over the links. This will be further discussed in Chapter 5.

The protocol to remove a real-time channel is straightforward. After all messages have been transmitted, the source node sends a channel-remove message to the NM which deletes the channel from its channel table. A message is also sent to the destination node informing it of the end of transmission.

With a centralized channel establishment protocol, all real-time channels are established by a single node containing the NM. This relieves all other nodes from the complex channel establishment functions, thus making it easy to implement and maintain. New channel establishment algorithms can be installed without disturbing all other nodes in the network. Other functions like accounting can be easily accommodated. Problems with the centralized protocol is its speed and reliability. Real-time channels have to be established one at a time, and each channel establishment needs to exchange several messages between the requesting node and the NM. Thus a node may have to wait a long time before its channel request is handled. Failure of the NM will cause the rejection of all channel requests.

The channel establishment can also be done in a distributed manner.

Protocol 3.4 (Distributed channel establishment) .

Step 1: To establish a real-time channel, the source node selects a route to the destination over which the real-time channel is to be established. It then divides the end-to-end delay bound D into link delay bounds d_i's satisfying Theorem 3.1 and sends a channel-request message which contains the channel parameters (T, M, D), ℓ_i's and d_i's, source and destination addresses to the first node on the route.

Step 2: Each intermediate node, after receiving a channel-request message, checks the schedulability of the channels over the requested outgoing link using Theorem 2.7. If the result is positive, the channel being established is registered in the node's channel table, and the channel-request message is forwarded to the next node on the route. Otherwise, a channel-reject message is sent back to the source node, and the channel is removed from the channel tables of the previous nodes.

Step 3: If a destination receives the channel-request message, it sends a channel-accept message to the source node. The source node can start message transmission using Protocol 3.1.

The reason for using M1 to calculate the link delay bounds in the above protocol is to avoid channel blocking. This distributed protocol allows several channel establishment processes to go on simultaneously. If M2 is used, then a real-time channel τ will first be established with the minimum link delay bound that each link of the route can provide. Before the link delay bounds are relaxed, as done in Step 2.3 of Protocol 3.3, it would be very difficult to establish other channels over these links since the unnecessary small link delay bounds have been assigned to channel τ. This channel blocking phenomenon could increase the rejection probability of a channel request.

As to the routing scheme of the channel establishment protocol, we recommend the minimum-hop routing. The reason is that the fewer links a real-time channel uses, the more likely the channel can be established since the end-to-end delay increases with the number of links. Also, the fewer links a real-time channel uses, the less effect it has on the establishment of future channels. Thus, in general, the minimum-hop routing can maximize a network's ability of supporting real-time channels. This fact will be discussed further in Chapter 5.

In spite of its many advantages, the real-time channel approach does have its limitations. One such limitation is the requirement of channel establishment before message transmissions. The channel establishment may take a time or even be rejected when the network is heavily loaded. This limitation makes real-time channels applicable to only those applications whose starting times can be delayed and whose communication sessions last for a relatively long period of time. Typical examples of such applications are interactive voice/video transmissions where it is acceptable for the users requesting new channels to receive "busy" signals but try the connections later. Once a connection is established, the life time is usually much longer than the channel establishment time. For distributed control/manufacturing systems, the communication channels between sensors/actuators and controllers are usually valid until the system changes. These channels should be appropriately established before the system starts operation. Failure to establish necessary communication channels means that the system is ill-designed. Real-time channels are not useful for the transmissions of short randomly-generated urgent messages, for which the long channel establishment time is usually intolerable and the delay of message transmission is not allowed. Enhancement of real-time channels to support such applications could be done in the following two ways.

1. For applications like alarm signal transmission, the source and destination nodes of communication channels are usually known in advance. Then, appropriate real-time channels can be established before the generation of messages. In this way, a randomly generated real-time message like an alarm signal can be transmitted over a pre-established real-time channel immediately after its generation. The problem with this approach is the inefficient use of the pre-established real-time channels since they are idle for the most of time.

2. If the source and destination nodes are not known in advance, a special real-time channel which goes over every link of the network can be established in advance. The link delay bound of this special channel can be set very large in order not to affect the establishment of other real-time channels. All randomly generated real-time messages are delivered over this special channel using any routing algorithm. In this way, real-time messages will be delivered with higher priority than that of non real-time messages, and transmissions of these messages will not affect the quality guarantees of

other real–time channels. However, there will be no delay bound guarantee for these randomly generated message and they are delivered on a best-effort basis.

3.4 Conclusion

The real–time channel approach is presented in this chapter with which the heterogeneous real–time communications in point-to-point computer networks can be flexibly supported. Real–time channels enable us to use a new transfer mode which has the advantages of both the circuit–switched (real–time communication) and the packet–switched (high transmission efficiency) transmission. In the following chapters, we will discuss the implementation, enhancement, and applications of real–time channels.

CHAPTER 4

IMPLEMENTATION ISSUES

4.1 Introduction

A key problem to the implementation of real–time channels is the existence of a fast switching node. Here by "fast" we mean that the switching/processing time of a packet can be controlled under the packet transmission time.

There has been extensive research on fast switch design [17, 18, 19, 20, 21]. With an output–buffer architecture, switches supporting Gbps transmission links have been shown to be feasible with the available VLSI technology [22]. However, to implement real–time channels, a switch must perform an extra function: scheduling of packet transmissions.

Unlike the conventional First-In-First-Out (FIFO) scheduling for which a newly arrived packet is simply put at the end of a waiting queue at an outgoing link and a transmitter always picks the first packet in the waiting queue to transmit, the deadline scheduling requires that either the transmitter searches a waiting queue to find a packet with the earliest deadline to transmit, or a newly arrived packet is inserted at a proper position such that a waiting queue is ordered such that the packet with the earliest deadline is always at the head of the queue. In either case, the whole waiting queue must be searched once for every packet transmitted. Previous research on real–time channels simply ignored the scheduling time by assuming a fast network processor which performs the packet insertion/deletion in software [23]. However, for a communication network with high link bandwidth (say larger than 100Mbps) and short packet size (say less than 1000 bits), the transmission time of a packet is of the order of microseconds. This makes the above assumption impractical.

If packet scheduling time cannot be controlled under packet transmission time, a queue will be formed at the scheduler. One should *not* use deadline scheduling for the queue at the scheduler since this would need another deadline scheduler. So, only the conventional FIFO or round robin scheduling can be used, which could result in a larger queuing delay at the scheduler than that at a transmission link, making the use of deadline scheduling at a transmission link meaningless. Using scheduling policies other than the deadline scheduling makes real–time channels lose the second advantage (channel protection) as discussed in Chapter 1. So, packet scheduling times must be reduced as much as possible to implement real–time channels.

Realizing the need for a fast deadline scheduler, a hardware implementation of it is proposed in this chapter which can schedule an incoming packet in at most 12 clock cycles (with the conventional software implementation, a couple of hundred clock cycles is not unusual). The design is verified and evaluated with the VERILOG hardware description language. Simulations showed enormous improvements over packet delays by reducing the scheduling time. We also compared the conventional FIFO and priority scheduling with

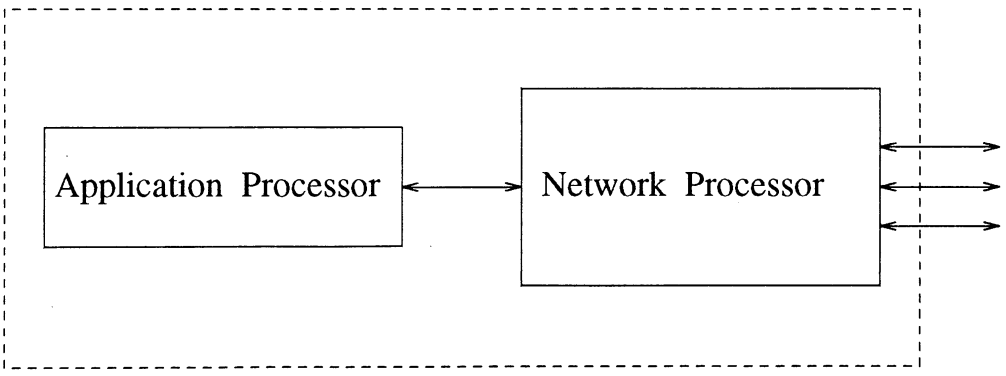

Figure 4.1: Node architecture

the deadline scheduling and showed the superiority of the latter in supporting real–time channels.

This chapter is organized as follows. Section 4.2 presents a detailed design of a deadline scheduler, which is then verified and evaluated in Section 4.3 by simulations. The chapter concludes with Section 4.4.

4.2 Design of a Fast Deadline Scheduler

The architecture of each node to be considered is shown in Fig. 4.1. Each node is composed of one or more application processors and a network processor. Application processors execute application programs and the network processors provide an interface for the application processors to communicate with application processors in other nodes. As discussed in [24], the usage of network processors can offload communication overhead from the application processors and increase parallelism between computation and communication.

The main functions of a conventional network processor are packet processing and routing. It also buffers incoming packets when there are contentions at outgoing links. Real–time channels require a network processor to perform two extra functions: deadline calculation (Protocol 3.1 or Protocol 3.2) and packet scheduling. The architecture of a network processor is shown in Fig. 4.2.

As shown in Fig. 4.2, incoming packets are first processed by the Routing Controller (RC), which performs routing, deadline calculation, error checking and other necessary processing. The RC then stores the packets in the memories of the corresponding outgoing links. The packet Scheduling Information (SI) which contains the channel ID, packet deadline, and the address of the packet in the memory is passed to the scheduler.

With the network processor architecture described above, the function of the scheduler is then accepting packet SIs from the RC, arranging them according to their deadlines and outputting the head of the queue to the transmitter when requested. The main operation involved here is to arrange the SIs into a priority queue where the SI of the packet with the earliest deadline is always at the head of the queue. When the number of packets buffered in the network processor is large, searching the priority queue for an appropriate position to insert a new SI could be quite time consuming if the scheduler were implemented in software.

The key component of our deadline scheduler is a hardware implementation of the priority queue into which a new SI can be inserted in two clock cycles. This is realized

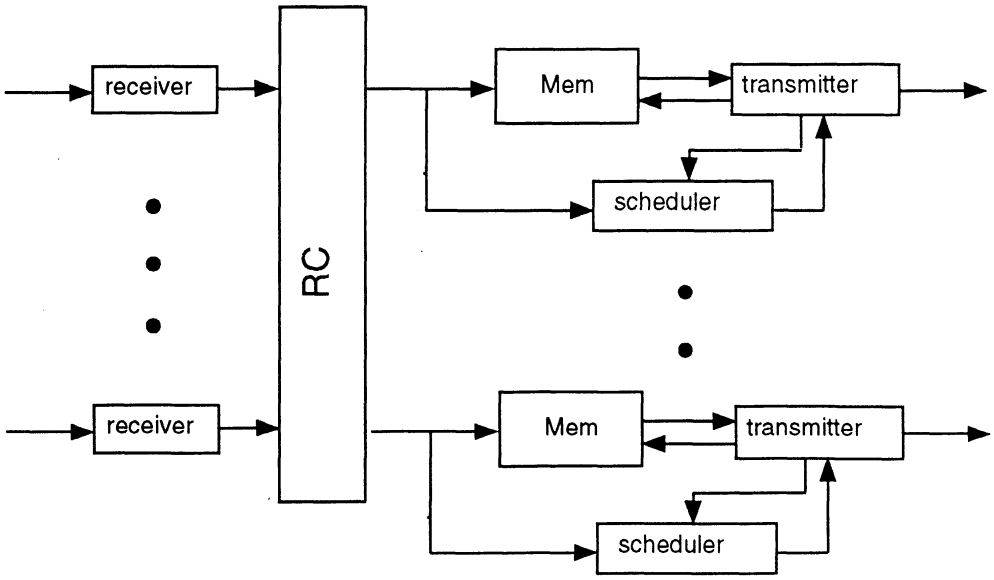

Figure 4.2: Network processor architecture

by assigning a comparator to each storage cell and comparing all elements stored in the priority queue with the new one simultaneously. Needless to say, such an implementation is very expensive and the size of the priority queue must be kept under a certain limit.

We manage to keep the queue size to the maximum number of real–time channels to be accommodated by an outgoing link. This is possible since the packets of a single real–time channel are always transmitted on a FIFO basis. Hence, the priority queue needs to hold at most one packet SI for each channel. When this packet is transmitted, the second packet SI of the channel (if there is one waiting for transmission) gets inserted in the priority queue.

The detailed scheme is shown in Fig. 4.3 – 4.5. There are two types of events that a deadline scheduler must handle: packet arrival and packet departure. On the arrival of a packet, the scheduler stores the SI of the packet in a register R1, it then checks if the priority queue (PQ) has a packet PI of the same channel (this information is stored in memory M1). If not, the SI of the arriving packet is inserted in PQ. Otherwise, they are stored in memory M2. On the departure of a packet, i.e, when the transmitter finishes transmitting a packet and requests the next one, the scheduler outputs the head of PQ and checks if M2 has a packet SI of the same channel (again, this information is stored in M1). If yes, the packet is fetched from M2 and inserted into PQ. Otherwise, it changes the contents in M1 to indicate that PQ no longer contains a packet SI of the channel. Since the arrival and departure of packets can happen simultaneously, care must be taken so that such collisions are handled consistently.

The scheduler is connected to the routing controller (RC) via a 32-bit data bus and a signal line IN. When the RC finishes processing an incoming packet, it asserts the IN line to the scheduler of the corresponding outgoing link. In the next two clock cycles, the PRC sends two data units DATA1 and DATA2 over the bus and then resets IN. The formats of DATA1 and DATA2 are shown in Fig. 4.6. They contain the channel ID, the deadline of the packet, and the address in the memory at which the packet is stored. The scheduler ignores all the following data transmitted over the bus (i.e., the packet itself to be stored in memory) until the IN line is asserted again.

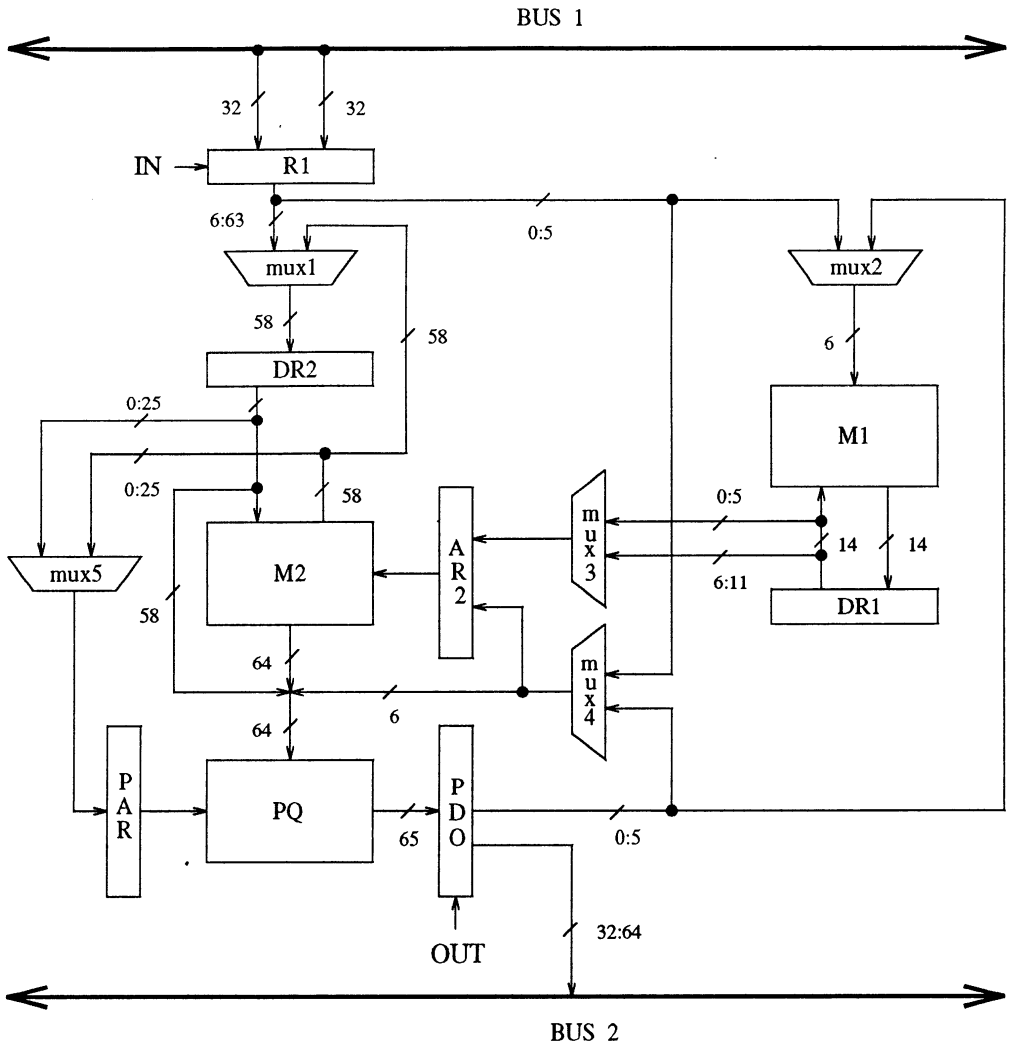

Figure 4.3: Scheduler data path

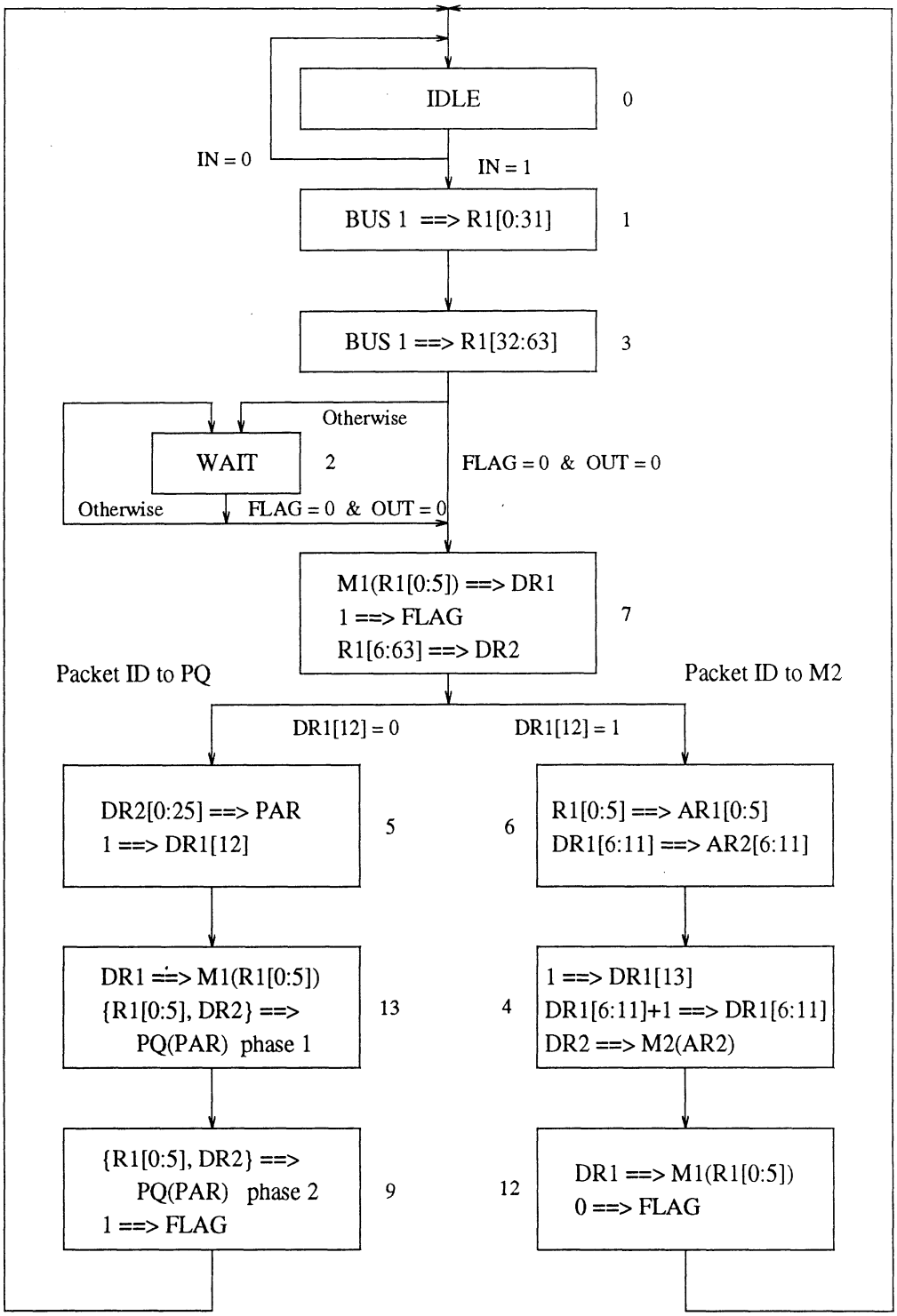

Figure 4.4: Flow graph of scheduling an incoming packet

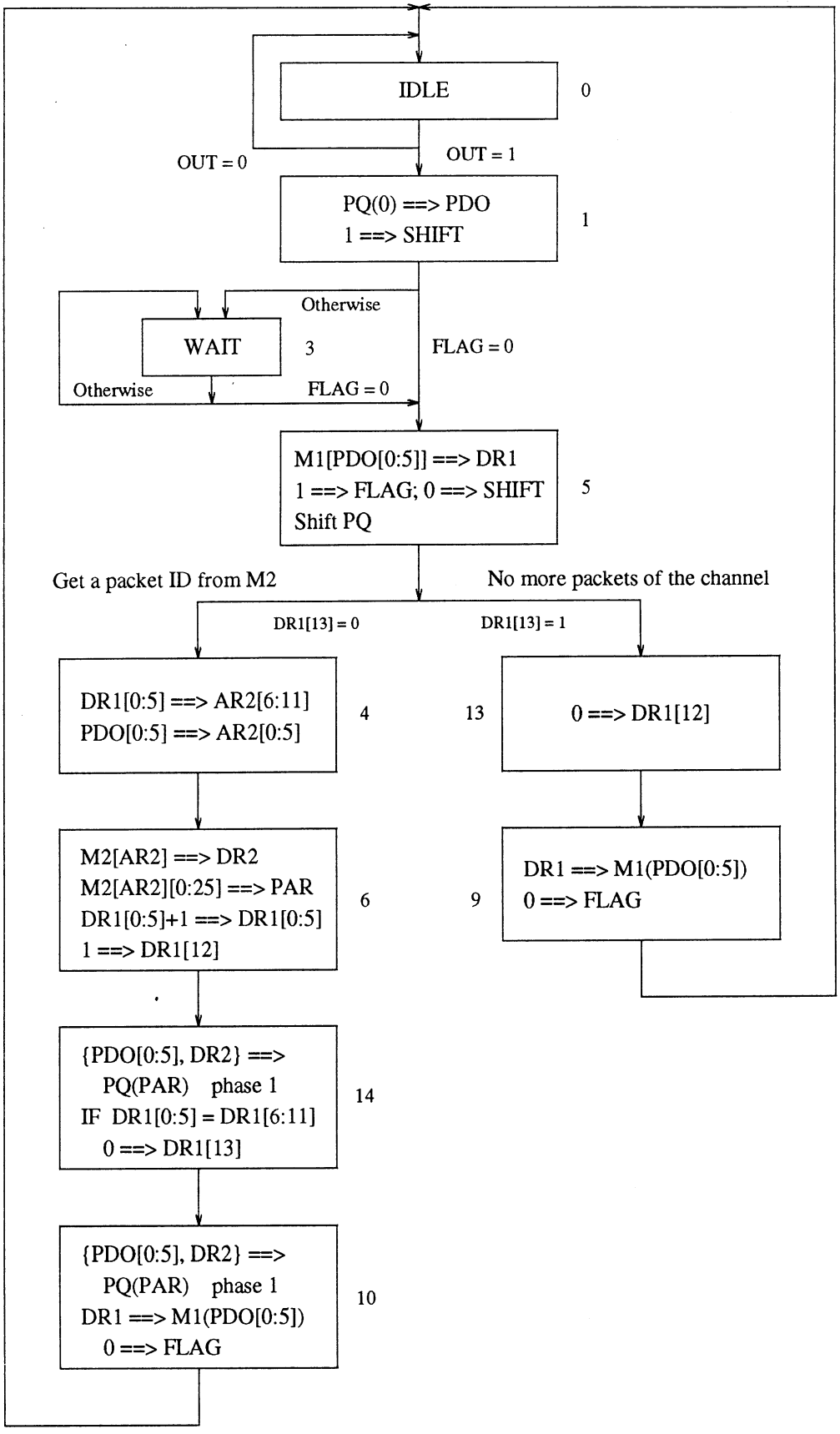

Figure 4.5: Flow graph of outputting packet

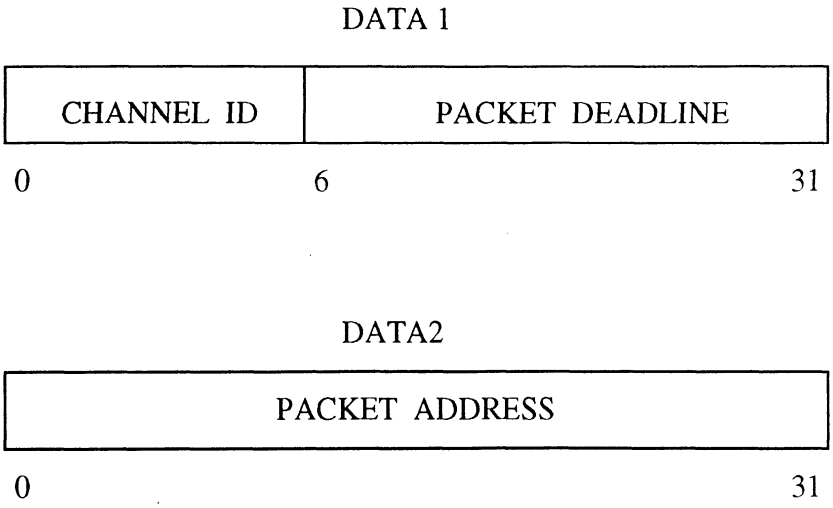

Figure 4.6: Data format

The interface of the scheduler to the transmitter is an output register PDO and an OUT line from the transmitter. The last (65th) bit of PDO is a valid bit which serves as a READY signal indicating if there are packets waiting to be transmitted. When the transmitter is idle and detects the READY signal, it reads the packet address from PDO and fetches the packet from the memory to transmit. The transmitter also asserts the OUT line for a couple of clock cycles to start the packet departure operations of the scheduler.

Insertion to PQ is composed of two phases (thus needs two clock cycles): comparison and insertion. In the first phase, the deadline of the new packet (stored in the register PAR) is compared with the deadlines of all packets stored in PQ, and the position the new packet should be inserted is found. The new packet is inserted in PQ in the second phase. The output of PQ is simply a shift right operation.

The scheduler can accept or output a packet SI at any time. However, the operations involved in the packet arrival or departure must be coordinated to avoid simultaneous requests for the memories and registers in the scheduler. This is achieved by adding two control bits FLAG and SHIFT.

As shown in Figs. 4.4 and 4.5, the FLAG bit allows only one operation (packet arrival or packet departure) to access the critical section. When the two operations try to enter the critical section at the same time, the access right is given to the packet departure operation by having the packet arrival operation check the OUT signal also. The SHIFT bit controls the shift operation of PQ. As discussed above, when PQ outputs a packet, it shifts right. However, this shift operation can not be done when the packet arrival operation is in the critical section. Thus, the SHIFT bit tells PQ not to insert the new packet at the head of the queue otherwise it will be mistakenly shifted out.

4.3 Verification and Evaluation

The verification and evaluation of the obtained design were done using the VERILOG hardware description language. We wrote two levels of descriptions, one functional that describes the interfaces to the Routing Controller (RC) and the transmitter, and the internal datapath of the scheduler without consideration of the actual logic circuit generating the control signals, and one structural that describes the real circuit of the scheduler. The functional description was used to perform high level performance evaluation, and the

structural description to verify the correctness of the obtained design. The reason for using two level simulation is that the performance evaluation does not need the details of the control logic, and the program for functional description runs faster than that for structural description since it does not need to simulate the control logic. A structural description is necessary since we have to prove the correctness of the design.

To verify the correctness of the design, we inputted a set of packets to the scheduler and checked if the internal state transients and sequences of the outputting packets were correct. A typical run of the structural description program is shown below, where the first value of the state represents the state of the packet arrival operation as shown in Fig. 4.4 and the second value the state of the packet departure operation as shown in Fig. 4.5.

```
VERILOG-XL 1.5c   Dec 17, 1991  16:34:12
  * Copyright Cadence Design Systems, Inc. 1985, 1988.    *
  *     All Rights Reserved.       Licensed Software.     *
  * Confidential and proprietary information which is the *
  *      property of Cadence Design Systems, Inc.         *
Compiling source file "test.v"
Highest level modules:
top
annotate

clock cycle = 20
time = 0   state = (0, 0)
time = 10   state = (1, 0)

Packet arrives at 30  channel ID = 1  deadline = 5500
time = 30   state = (3, 0)
time = 50   state = (7, 0)
time = 70   state = (5, 0)
time = 90   state = (13, 0)
time = 110   state = (9, 0)
time = 130   state = (0, 1)

Packet departs at 150  channel ID = 1  deadline = 5500
time = 150   state = (0, 5)
time = 170   state = (0, 13)
time = 190   state = (0, 9)
time = 210   state = (0, 0)
time = 230   state = (1, 0)

Packet arrives at 250  channel ID = 2  deadline = 7500
time = 250   state = (3, 0)
time = 270   state = (7, 0)
time = 290   state = (5, 0)
time = 310   state = (13, 0)
time = 330   state = (9, 0)
time = 350   state = (0, 0)
```

```
time = 450   state = (1, 0)

Packet arrives at 470  channel ID = 0  deadline = 3500
time = 470   state = (3, 0)
time = 490   state = (7, 0)
time = 510   state = (5, 0)
time = 530   state = (13, 0)
time = 550   state = (9, 0)
time = 570   state = (0, 0)
time = 2130   state = (1, 0)

Packet arrives at 2150  channel ID = 3  deadline = 11610
time = 2150   state = (3, 1)

Packet departs at 2170  channel ID = 0  deadline = 3500
time = 2170   state = (2, 5)
time = 2190   state = (2, 13)
time = 2210   state = (2, 9)
time = 2230   state = (7, 0)
time = 2250   state = (5, 0)
time = 2270   state = (13, 0)
time = 2290   state = (9, 0)
time = 2310   state = (0, 0)
time = 4110   state = (1, 0)

Packet arrives at 4130  channel ID = 4  deadline = 15600
time = 4130   state = (3, 0)
time = 4150   state = (7, 0)
time = 4170   state = (5, 1)

Packet departs at 4190  channel ID = 2  deadline = 7500
time = 4190   state = (13, 3)
time = 4210   state = (9, 3)
time = 4230   state = (0, 5)
time = 4250   state = (0, 13)
time = 4270   state = (0, 9)
time = 4290   state = (0, 0)
time = 5130   state = (1, 0)

Packet arrives at 5150  channel ID = 3  deadline = 21610
time = 5150   state = (3, 0)
time = 5170   state = (7, 0)
time = 5190   state = (6, 0)
time = 5210   state = (4, 0)
time = 5230   state = (12, 0)
time = 5250   state = (0, 0)
time = 6190   state = (0, 1)
```

```
Packet departs at 6210  channel ID = 3  deadline = 11610
time = 6210   state = (0, 5)
time = 6230   state = (0, 4)
time = 6250   state = (0, 6)
time = 6270   state = (0, 14)
time = 6290   state = (0, 10)
time = 6310   state = (0, 0)
L848 "test.v": $finish at simulation time 8000
370486 simulation events
CPU time: 0 secs to compile + 1 secs to link + 13 secs in simulation
End of VERILOG-XL 1.5c   Dec 17, 1991  16:34:28
```

At time 30, the first packet arrived at the scheduler. After a series of state transients, the scheduler reached the state $(9, 0)$ at time 110, where the header of the packet was inserted at the priority queue (PQ). The transmitter fetched the pointer of the packet at the next clock cycle (time 130) and the packet got transmitted at time 150.

The minimum packet scheduling time is thus $(150 - 30)/20 = 6$ clock cycles. In other words, if the output link is idle, a cut–through packet experiences the delay of 6 clock cycles at the scheduler. From the flowgraphs of the scheduler (Figs. 4.4 and 4.5), the maximum cycles needed to output a packet is 6, so the maximum scheduling time is $6 + 6 = 12$ clock cycles when a packet arrival and a packet departure occur simultaneously.

Competition for the critical section occurred at times 2170 and 4190. In the former case, the packet arrival operation waited at state 2 for three cycles until the packet departure operation finished. In the latter case, the packet departure operation waited at state 3 for two cycles until the packet arrival operation finished. At time 5150, the state transitions showed that the incoming packet was stored at M1, and this packet was inserted in PQ after a packet of the same channel got transmitted at time 6210. It is also easy to see that the packets got transmitted with the correct order, i.e., the deadline scheduling was implemented.

Performance evaluations were done using the functional description program. Five channels were created each of them had inter-arrival packet generation times uniformly distributed between 500 and 750 clock cycles. Packet transmission time was 100 clock cycles. With a 10MHz clock, these figures correspond to a 100Mbps transmission link, and a 1000-bit packet size.

The delay of a packet is defined as the time period between its generation and when the transmitter starts transmitting it. The maximum packet delay of each channel was measured by simulating the system for 10 seconds (or 200,000,000 clock cycles), which took about 10 hours to run the functional description program on a SUN Spark workstation. Theoretically, the simulation time should be as long as possible to reveal the worst case packet delays.

The first experiment investigated the effects of scheduling times on the maximum packet delays to verify our claim for the need of a fast deadline scheduler. The measured maximum packet delays of the channels with different scheduling times were plotted in Fig. 4.7. The dashed lines in the figure represent the requested delay bounds of the channels.

At a scheduling time of 12 clock cycles, all the channels met the requested delay bounds. The real–time channels successfully supported channels with different delay requirements. However, as the scheduling time increased to 30 clock cycles, packets of channel 1

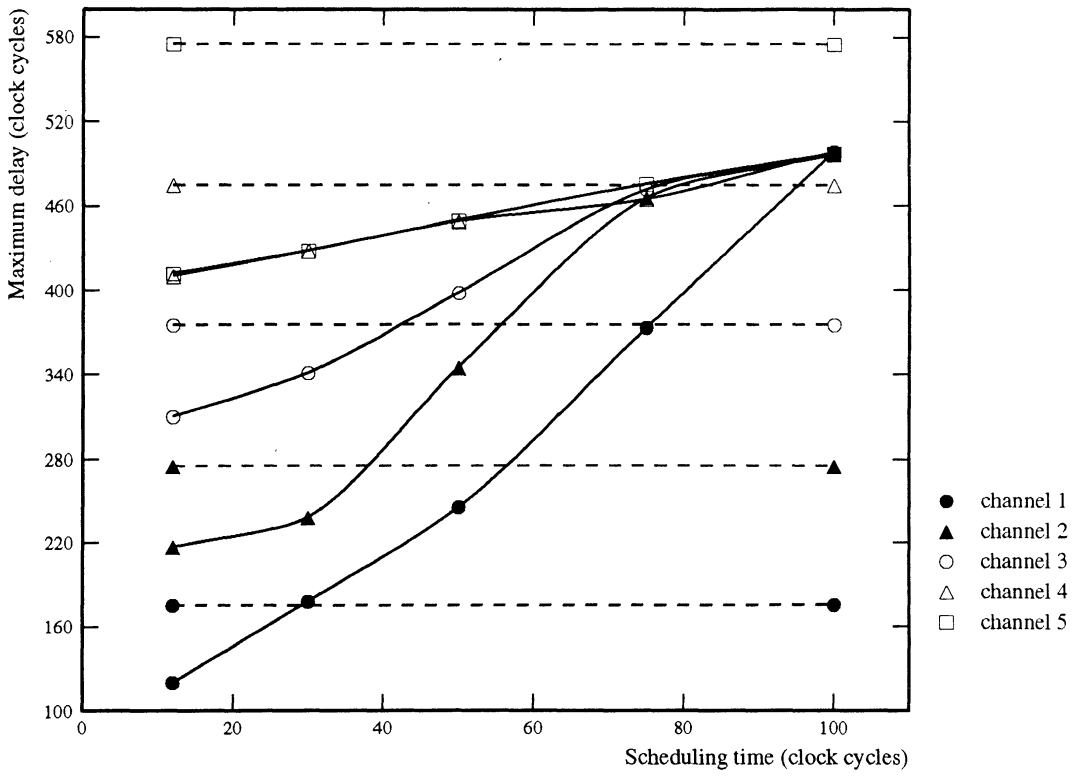

Figure 4.7: Effects of scheduling times

missed their deadlines. With a further increase of the scheduling time, all channels tend to have the same maximum packet delay, which means that the advantages of deadline scheduling disappeared completely. Actually, as will be shown by the next experiment, the queuing delays at the scheduler resulted larger packet delays than that with FIFO scheduling. The results of this experiment clearly showed the importance of a fast deadline scheduler in the implementation real–time channels.

The second experiment compared the deadline scheduling with the FIFO scheduling. The scheduling times for the deadline scheduling and the FIFO scheduling are assumed to 12 clock cycles and 2 clock cycles, respectively. The results are shown in Fig. 4.8, where the dashed lines represent requested delay bounds of the real–time channels. It is easy to see that a FIFO scheduler treats all packets equally therefore is not appropriate for accommodating real–time channels where different channels may have different delay requirements.

The third experiment compared deadline scheduling with priority scheduling. With a priority scheduler, an urgent channel is assigned a high priority, thus its packets are more likely to meet the delay bound than with the FIFO scheduling. However, a problem with priority scheduling is that the channels are not protected from each other. A higher priority channel can easily make lower priority channels miss their deadlines. Fig. 4.9 shows the situation when channel 1 (with the highest priority) violates its traffic specification and generates packets at a higher rate (generate a packet every 250 – 500 clock cycles instead of the specified 500 – 750 clock cycles). With deadline scheduling, only the packets of the violating channel missed their deadlines. But with priority scheduling, the violating channel made three other channels miss their deadlines. This error diffusion phenomenon is very

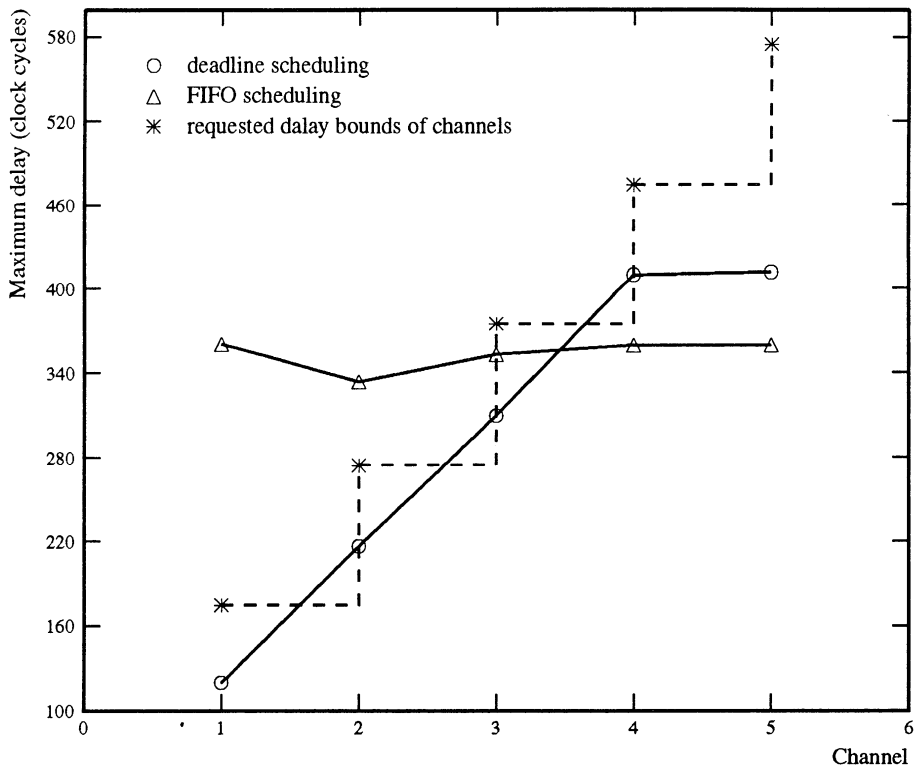

Figure 4.8: Comparison of deadline scheduling with FIFO scheduling

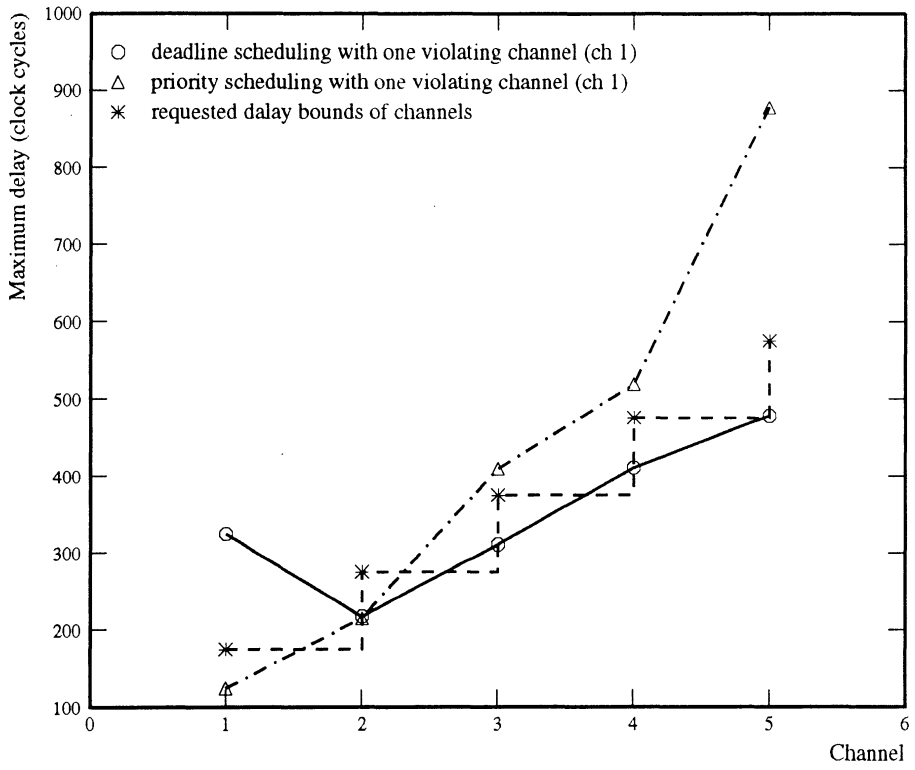

Figure 4.9: Comparison of deadline scheduling with priority scheduling

undesirable for real–time systems.

4.4 Conclusion

In this chapter, we have showed the importance of a fast deadline scheduler for real–time channels and established the feasibility of implementing such a device. The experiments performed have clearly underscored the superiority of deadline scheduling over FIFO and priority scheduling in this domain, and also shown that for high packet bandwidths and slow scheduling, the scheduler becomes the bottleneck and deadline scheduling losses its superiority.

CHAPTER 5

FAULT–TOLERANT REAL–TIME CHANNELS

5.1 Introduction

Fault–tolerant and real–time communication is very important yet difficult to achieve. Traditional protocols like the TCP achieve reliable communication through acknowledgment and retransmission schemes, where one gains the reliability at the cost of timeliness. In this chapter, we discuss how this problem can be solved by using real–time channels and exploring the spatial redundancy of a given network topology.

The real–time channels discussed so far are *not* fault–tolerant. All messages of a channel are transmitted along a static path, so a single component failure could disable the entire channel. A natural way to increase the reliability of a real-time channel is to expand the channel with some extra links and nodes such that packets can be re-routed around faulty components on the original channel. One extreme is to use all links and nodes in the network such that packets can be successfully routed in a timely manner as long as the network remains connected. However, this method is usually very expensive. The cost comes not only from the large number of links/nodes involved, but also from the delay bound requirements over the links. For a real-time channel which remains operational as long as the network is connected, the number of hops a packet may traverse in the worst case is very large. The large number of hops from the source to destination node requires each link on the path to provide a very small delay bound so that the end–to–end delay bound can be guaranteed. From Theorem 2.6, it is easy to see that requests establishing real–time channels with very small requested delay bounds over a link are very likely to be rejected.

By making a tradeoff between reliability and cost, we first in Section 5.2 establish *Single Failure Immune* (SFI) real–time channels which guarantee the timely delivery of packets as long as each packet encounters no more than one link/node failure on its way to the destination node. In Section 5.3, we discuss how a class of more reliable real–time channels, called the *Isolated Failure Immune* (IFI) real–time channels, can be established in networks with the hexagonal mesh topology. For a real–time communication system that can tolerate rare short–period break downs, we discuss in Section 5.4 how backup channels can be established which is a less expensive way to increase the reliability of real–time channels. This method is also applicable to cases where a SFI circuit cannot be established due to the poor connectivity of the network. The chapter concludes with Section 5.5.

5.2 Single Failure Immune Real–time Channels

For convenience of presentation, we introduce some definitions first. A communication network is modeled as a directed graph $N = \{V, E\}$, where V is a set of nodes and

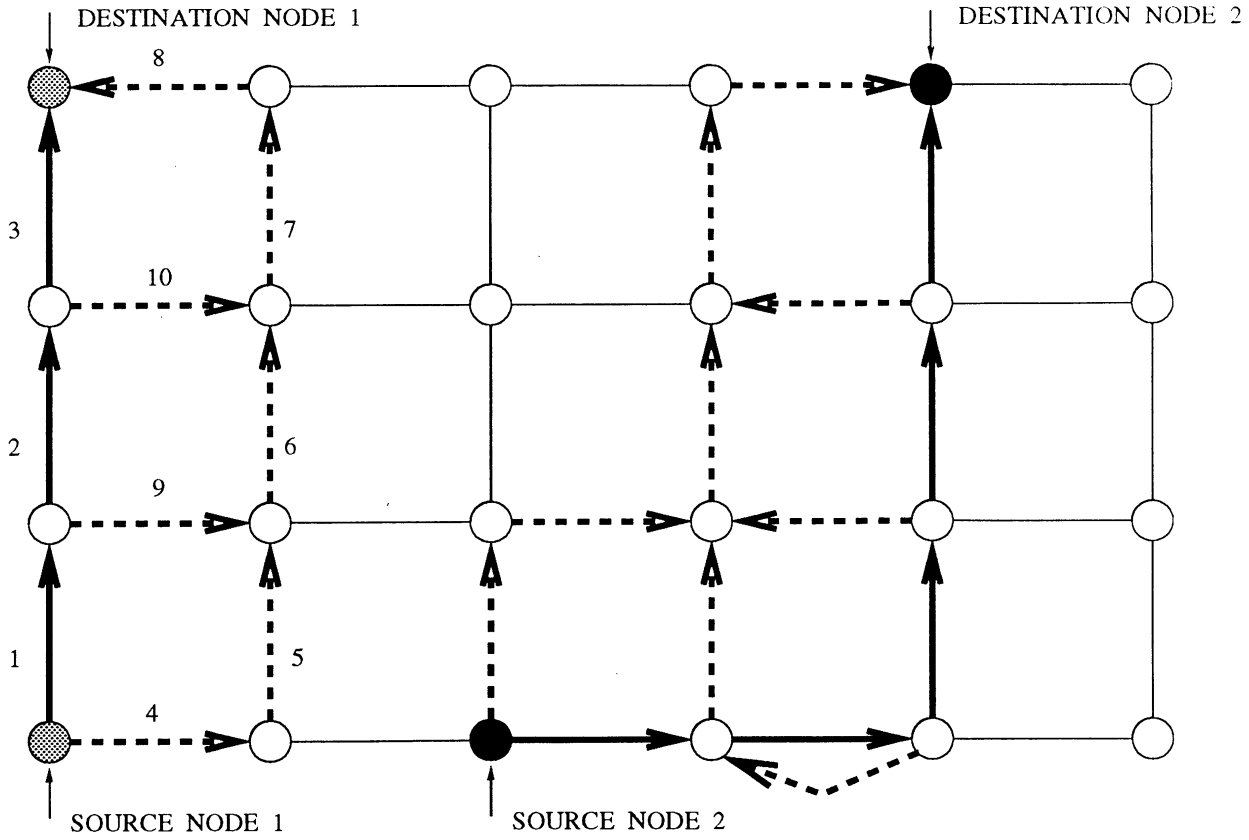

Figure 5.1: Two optimal SFI circuits in a mesh network.

E is a set of directed links. A *basic circuit* from node v_0 to node v_k in a network is defined as a sequence $C_b = v_0 e_1 v_1 e_2 \ldots e_k v_k$, where v_i's are nodes and $e_i = \overrightarrow{v_{i-1} v_i}$ is a directed link from v_{i-1} to v_i.

Definition 5.1 (Single Failure Immune (SFI) circuit) .
A SFI circuit, denoted by C_s, from node v_0 to node v_k is defined as a basic circuit $C_b = v_0 e_i v_1 e_2 \cdots e_k v_k$ augmented with some extra nodes and links, which are called the detour *of C_b, such that when node v_i $(1 \leq i < k)$ or link e_i $(1 \leq i \leq k)$ is removed from C_b, there exists a basic circuit from v_{i-1} to v_k in the remaining C_s.*

With the above definition, when a packet arrives at v_{i-1} and finds that either v_i or e_i is faulty, it can always be re-routed over a detour link leading to the destination node. Thus a SFI circuit guarantees that a packet will always be delivered to the destination node as long as no more than one link or node (except the source and destination nodes) is faulty. Figure 5.1 shows two SFI circuits in a mesh network, where the solid arrows represent the basic circuits and the dashed arrows represent the detour.

To establish a SFI real–time channel, the first step is to find a SFI circuit on which the real–time channel is to be established. A straightforward way to find a SFI circuit from v_0 to v_k is given as follows. First, select a basic circuit $C_b = v_0 e_1 \cdots, v_k$. Then, for $i = 1, \cdots k$, remove v_i (e_i if $i = k$) from C_b and establish a basic circuit from v_{i-1} to v_k in the remaining network. Clearly, the union of all the basic circuits forms a SFI circuit from v_0 to v_k. Failure of this algorithm means that no SFI circuits exist with the selected C_b.

If a SFI circuit is to be used for the establishment of a SFI real–time channel. some extra features are desirable. From Theorem 2.6, we see that an established real–time channel over a link affects the link's ability to accommodate future channels. In establishing a new channel, it is thus desirable to minimize this negative influence on future channels to be established.

There are two ways to measure this influence of a real–time channel. First, the more links a real–time channel traverses, the more pronounced the influence will become. This is because a larger number of links are involved in establishing a long channel than a short channel. Secondly, over a single link, the smaller the link delay bound the more the influence is (see Theorem 2.6). Thus, to reduce a real–time channel's influence on the network's ability to establish future channels, one should run it through as few links as possible, and make the requested packet delivery delay over each link as large as possible. Note that the second objective is consistent with the first, because the more links a real–time channel traverses, the smaller the requested packet delivery delay per link would become. Hence, the minimum–hop routes are best suited for real–time channels.

Another advantage of minimum–hop routing for real–time channels is the reduction of real–time packets' influence over non real–time packets. If each real–time packet traverses through a minimum number of links, the total real–time traffic in the network would be minimized. Since transmission priority is usually given to real–time packets over non real–time packets, minimizing real–time traffic effectively minimizes its influence on non real–time packets in the network.

For SFI real–time channels, the influence of the existing real–time channels on future channel establishment also includes that over detour paths. Thus, one should establish a SFI real–time channel over a SFI circuit which needs a minimum number of extra links.

In summary, we have the following two goals in selecting a SFI circuit for the establishment of a SFI real–time channel:

G1: The basic circuit is a minimum–hop route from the source to the destination node. In the case of link/node break down, the detour should also be the minimum–hop routes in the remaining network.

G2: Under the constraint of G1, the total number of links on the detour of a SFI circuit should be as small as possible.

A SFI circuit is said to be *optimal* if it achieves the above two goals. It is not difficult to find an optimal SFI circuit in some widely–used regular networks like meshes and hypercubes. Figure 5.1 gives an example of two optimal SFI circuits in a mesh network.

For an arbitrary–topology network, however, the optimal SFI circuits are not always readily obtainable. The difficulty comes from the existence of multiple minimum–hop basic circuits between two nodes in a network. Different choices of the basic circuit and detour could result in different numbers of extra links needed for a SFI circuit. We propose a heuristic algorithm for finding a SFI circuit as follows.

Algorithm 5.1 (Construction of SFI circuits) .

Step 1. Set up a minimum–hop basic circuit $C_b = v_0 e_1 v_1 \cdots e_k v_k$ from the source node v_0 to the destination node v_k.

Step 2. Initialize the set of extra nodes and links $C := \emptyset$. For $i = 1, \ldots, k$, do the following.

Step 2.1. Remove from the original network node v_i if $i < k$, and a link e_k if $i = k$.

Step 2.2. Establish a minimum-hop basic circuit C_i from v_{i-1} to v_k in the remaining network. At any node, if there are two directions both leading to a minimum-hop circuit, the one to a node which is closer to $C_b \cup C$ is selected. Break a tie using the following rules: (1) choose the one which does not introduce a new link, (2) choose the one which is closer to C_b, and (3) break the tie arbitrarily. If a basic circuit from v_0 to v_k does not exist, go to Step 3.

Step 2.3. Suppose C_i intersects node v_j in C. If there is node v_n in C_b such that (1) there is a link $\overrightarrow{v_j v_n}$ in C, (2) the basic circuit from v_j to v_k in C_i does not contain any node v_l, $l < n$, and (3) the number of hops from v_n to v_k in C_b plus 1 is not smaller than the number of hops from v_j to v_k in C_i, then remove $\overrightarrow{v_j v_n}$ from C. Update $C := C \cup (C_i/C_b)$.

Step 3. If the algorithm fails at Step 2, there does not exist a SFI circuit from v_0 to v_k. Otherwise, $C \cup C_b$ is a SFI circuit connecting v_0 and v_k with C_b as its basic circuit and C as the set of extra links and nodes.

The heuristic used in the algorithm is that a detour route C_i should be as close to the existing routes $C_b \cup C$ as possible. In this way, C_i will most likely intersect $C_b \cup C$, thus reducing the number of links needed. The purpose of Step 2.3 is to remove any redundant links and nodes. As an example, we show how a SFI circuit from SOURCE NODE 1 to DESTINATION NODE 1 in Figure 5.1 can be established using Algorithm 5.1. The relevant nodes and links and their labels are shown in Figure 5.2.

The sequences of the SFI circuit establishment steps are shown below.

Step 1. Set up a minimum-hop basic circuit $C_b = v_0 e_1 v_1 e_2 v_2 e_3 v_3$.

Step 2. Initialize the set of extra nodes and links $C := \emptyset$. $i = 1$.

Step 2.1 Remove node v_1 from the original network.

Step 2.2 Establish a minimum-hop basic circuit from v_0 to v_3 in the remaining network: $C_1 = v_0 e_4 v_4 e_5 v_5 e_6 v_7 e_{11} v_2 e_3 v_3$. At node v_6, both e_{11} and e_7 lead to a minimum-hop circuit, but e_{11} is chosen since it leads to a node closer to C_b.

Step 2.3 No nodes and links can be removed from C since $C = \emptyset$. Update $C := C \cup \{C_1/C_b\} = \{e_4, e_5, e_6, e_{11}, v_4, v_5, v_6\}$.

Step 2. $i = 2$.

Step 2.1 Remove node v_2 from the original network.

Step 2.2 Establish a minimum-hop basic circuit from v_1 to v_3 in the remaining network: $C_2 = v_1 e_9 v_5 e_6 v_6 e_7 v_7 e_8 v_3$.

Step 2.3 C_2 intersects node v_6 in C and there is node v_2 in C_b such that (1) e_{11} directs from v_6 to v_2, (2) the basic circuit from v_6 to v_3 in C_2 does not contain any node v_l, $l < 2$, and (3) the number of hops from v_2 to v_3 in C_b plus 1 equals the the number of hops from v_6 to v_3 in C_2. Thus, remove e_{11} from C_2. Update $C := C \cup \{C_2/C_b\} = \{e_4, e_5, e_6, e_7, e_8, e_9, v_4, v_5, v_6, v_7\}$.

Step 2. $i = 3$.

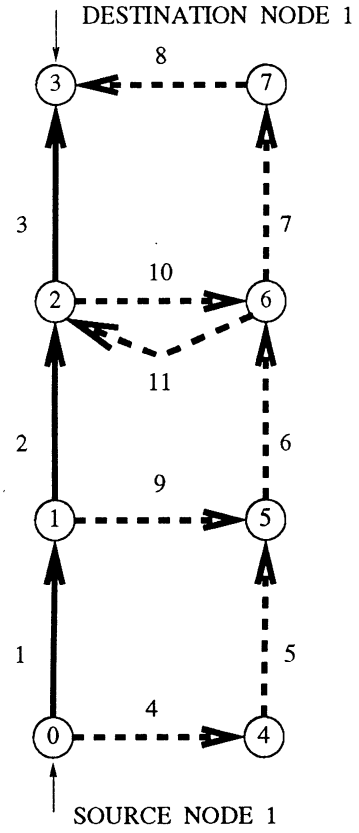

Figure 5.2: An example of using Algorithm 3

Step 2.1 Remove link e_3 from the original network.

Step 2.2 Establish a minimum-hop basic circuit from v_2 to v_3 in the remaining network: $C_3 = v_2 e_{10} v_6 e_7 v_7 e_8 v_3$.

Step 2.3 Update $C := C \cup \{C_3/C_b\} = \{e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, v_4, v_5, v_6, v_7\}$.

Step 3 Then, $C \cup C_b$ is a SFI circuit from v_0 to v_3 with C_b as its basic circuit and C as the set of extra links and nodes.

Comparing with Fig. 5.1, we see that an optimal SFI circuit is constructed using Algorithm 5.1. Although this is not always true, the algorithm performs relatively well for most networks.

We now develop algorithms for the establishment of a SFI real-time channel over a SFI circuit. Let $C_s = C \cup C_b$ be a SFI circuit on which a SFI real-time channel is to be established, where $C_b = v_0 e_1 v_1 \cdots e_k v_k$ is the basic circuit from the source node v_0 to the destination node v_k and C is the set of extra links and nodes. Suppose there are m links in C_s. Label the links in C as $e_{k+1}, \cdots, e_m$. Denote $C_0 = C_b$. For $i = 1, 2, \cdots, k$, let C_i be the minimum-hop circuit from v_0 to v_k in C_s when link v_i (e_k if $i = k$) is removed. Define a $(k+1) \times m$ circuit-link matrix $M = (m_{ij})_{k \times m}$ as follows:

$$m_{ij} = \begin{cases} 1 & \text{if } C_i \text{ contains } e_{j+1} \quad i = 0, \cdots, k, \\ 0 & \text{otherwise} \qquad\qquad\quad j = 0, \cdots, m-1 \end{cases}$$

Suppose link e_i guarantees a delay bound d_i, and let C_p denote the time needed to transmit a maximum–size packet. Then, from Theorem 3.1, a SFI real–time channel can be established with an end–to–end delay bound D if and only if the following delay inequality is satisfied:

$$M \begin{pmatrix} d_1 \\ \vdots \\ d_m \end{pmatrix} \leq \begin{pmatrix} D_1 \\ \vdots \\ D_m \end{pmatrix} \tag{5.1}$$

where

$$\begin{pmatrix} D_1 \\ \vdots \\ D_m \end{pmatrix} = \left(M \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} - \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right) \max\{0, (C - C_p)\} + \begin{pmatrix} D \\ \vdots \\ D \end{pmatrix} \tag{5.2}$$

Then, there are two ways to establish a SFI real–time channel over a SFI circuit.

Algorithm 5.2 (Establishment of SFI Real–time Channels (1)) .

Step 1. For $i = 1, 2, \cdots, m$, assign a link delay bound d_i over e_i, such that the inequality (5.1) is satisfied.

Step 2. Using Theorem 2.7, check the schedulability of the channel over each link. If all the checks are positive, the requested real–time channel can be established with the assigned link delays d_i's. Otherwise, the channel establishment request is rejected.

Algorithm 5.3 (Establishment of Real–time Channels (2)) .

Step 1. Using Theorem 2.8, calculate the minimum packet delay bound $d_{min,i}$ over link e_i, $i = 1, \cdots, k$.

Step 2. Set $d_i := d_{min,i}, i = 1, \cdots, m$. If the inequality (5.1) is satisfied, the requested real–time channel can be established. Assign the link delay over e_j to be $d_i =: d_{min,i} + \delta_i$, where δ_i's satisfy

$$M \begin{pmatrix} \delta_1 \\ \vdots \\ \delta_m \end{pmatrix} \leq \begin{pmatrix} D_1 \\ \vdots \\ D_m \end{pmatrix} - M \begin{pmatrix} d_{min,1} \\ \vdots \\ d_{min,m} \end{pmatrix} \tag{5.3}$$

Otherwise, the channel establishment request is rejected.

Since the inequality (5.3) can not determine a unique solution, we need to give a rule to choose one of the solutions. The reason for increasing the delay bound over link e_i from $d_{min,i}$ to $d_{min,i} + \delta_i$ in Algorithm 5.3 is to reduce the channel's influence on the link's ability to establish more real–time channels. The value of δ_i represents the degree of the influence reduced. Thus, with respect to a single link e_i, one should set δ_i as large as possible. However, since the maximizing of δ_i must be done under the constraint of inequality (5.3), increase of δ_i may cause the decrease of δ_j of another link e_j. To make the whole network evenly loaded, we use the following max-min rule to choose a solution from the inequality (5.3).

Max-min Rule: Among all solutions satisfying the inequality (5.3), choose the one whose smallest element, i.e., $\min_{1 \leq i \leq m} \delta_i$, has the maximum value. If there are more than one solution satisfying this rule, choose the one whose second smallest element has the maximum value. Repeat this process until a unique solution is obtained.

The max-min rule can be easily implemented with the following algorithm.

Algorithm 5.4 (Distribution of Link Delays) .

Step 1. Initialize the set of variables to be determined as $S = \{\delta_1, \cdots, \delta_m\}$.

Step 2. For all $\delta_i \in S$, replace δ_i with a single variable δ in inequality (5.3). Calculate the maximum value of δ satisfying inequality (5.3). Notice that Eq. (5.3) contains m inequalities, all elements of M are either 0 or 1, thus the maximum value of δ makes at least one inequality become an equality. Remove all δ_i's from S which are contained in the equality and set them to be the obtained maximum value of δ.

Step 3. If $S = \emptyset$, stop. Otherwise, goto Step 2.

As an example of using Algorithms 5.3 and 5.4, we show how a SFI real-time channel can be established over the SFI circuit from source node 1 to destination node 1 in Fig. 5.1. Suppose this real-time channel is to be established in an otherwise idle network and has the minimum message inter-arrival time $T = 100$, maximum message transmission time $C = 5$, and requested end-to-end message delivery delay bound $D = 60$. Suppose $C_p = C$. The labels of the links on the SFI circuit are shown in Fig. 5.1. Then, the circuit-link matrix is

$$M = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \tag{5.4}$$

From Theorem 2.8, we have $d_{min,i} = 5$ for $1 \leq i \leq 10$. Thus, the right-hand side of inequality (5.3) equals $(45\ 35\ 35\ 35)^T$. Using Algorithm 5.4, we first set $S := \{\delta_1, \cdots, \delta_{10}\}$. Replacing all elements in S with a single variable δ, inequality (5.3) becomes

$$\begin{pmatrix} 3 \\ 5 \\ 5 \\ 5 \end{pmatrix} \delta \leq \begin{pmatrix} 45 \\ 35 \\ 35 \\ 35 \end{pmatrix} \tag{5.5}$$

The maximum value δ is then 7 and with which the second, third and fourth inequalities of Eq. (5.3) become equalities which contain all δ_i's except δ_3. Thus, at the next iteration of Step 2 in Algorithm 5.4, $S = \{\delta_3\}$. Replacing δ_3 with δ and set all other variables in Eq. (5.3) to be 7, it becomes $14 + \delta \leq 45$. The maximum value of δ is 31.

Thus, the solution from algorithm 5.4 is $(\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_7, \delta_8, \delta_9, \delta_{10}) = (7, 7, 31, 7, 7, 7, 7, 7, 7, 7)$. Using Algorithm 5.3, the SFI real-time channel is established with link delay bounds $(d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10}) = (12, 12, 36, 12, 12, 12, 12, 12, 12, 12)$.

5.3 Isolated Failure Immune Real–time Channels

Making a real–time channel more robust than just tolerating a single failure is usually very difficult and requires reservation of significantly more network resources. We discuss in this section how the problem can be solved by choosing a proper network topology.

If a network has a wrapped hexagonal mesh topology [25], one can readily establish Isolated Failure Immune (IFI) real–time channels. An IFI real–time channel guarantees the timely delivery of messages in the presence of network component failures as long as the failures are *isolated* with respect to the channel. Node failures are said to be *isolated* with respect to a real–time channel if the source and destination nodes of the channel are not faulty and any two faulty nodes in the channel are not adjacent. Link failures (a link failure is caused by either the failure of the link itself or the failure of the node which the link leads to) are said to be *isolated* if any two faulty links are not originated from a same *non faulty* node or directed to the destination node. Figure 5.3 shows four types of non-isolated failures: (a) two faulty nodes which are adjacent, (b) two faulty links which originate from the same node, (c) same as (b) except that one link is made unusable (thus regarded faulty) by the failure of another node, and (d) two incoming faulty links of the destination node. Another two types of non-isolated failures are the failures of the source and destination nodes, respectively. Figure 5.4 shows an example of an IFI channel from node 1 to node 6 and one pattern of tolerable isolated failures.

The isolated failure immune communication problem for *undirected networks* was first discussed in [26] where the authors proved that a 2-tree[1] is a minimum IFI network. In other words, any IFI network must contain a spanning 2-tree. This result excludes almost all commonly–used network topologies (e.g., rings with more than 3 nodes, rectangular meshes, and hypercubes) from the candidate set of the IFI networks, except for the hexagonal mesh.

An IFI real–time channel has the following advantages over a basic real–time channel:

High Reliability: The channel can tolerate a large number of component failures as long as they are isolated. For example, the IFI channel shown in Figure 5.4 can tolerate as many as 7 faulty links and 2 faulty nodes, which represent 70% of the links and 33% of the nodes that the channel runs through.

Easy Failure Detection: Non-isolated failures in the network can be easily detected using only local information, i.e., the status of a node's own links and its neighbor nodes. This makes the system maintenance extremely easy. A node can safely shut down one of its links or itself by checking the status of its links and the neighboring nodes.

Transmission of Emergency Messages: Notice that a path between any pair of nodes in a network can always be constructed using only those links whose failure will not cause non isolated failures. So, in the absence of network component failures, an emergency message can always be transmitted from a source node to a destination node using the full link bandwidth on its path without interrupting existing real–time channels.

At the Real–Time Computing Laboratory, the University of Michigan, we are currently building an experimental distributed real–time fault–tolerant systems called HARTS

[1] A 2-tree can be constructed as follows. Two nodes connected by a link is a 2-tree. A new node can be added to a 2-tree by connecting it to two neighboring nodes in the 2-tree.

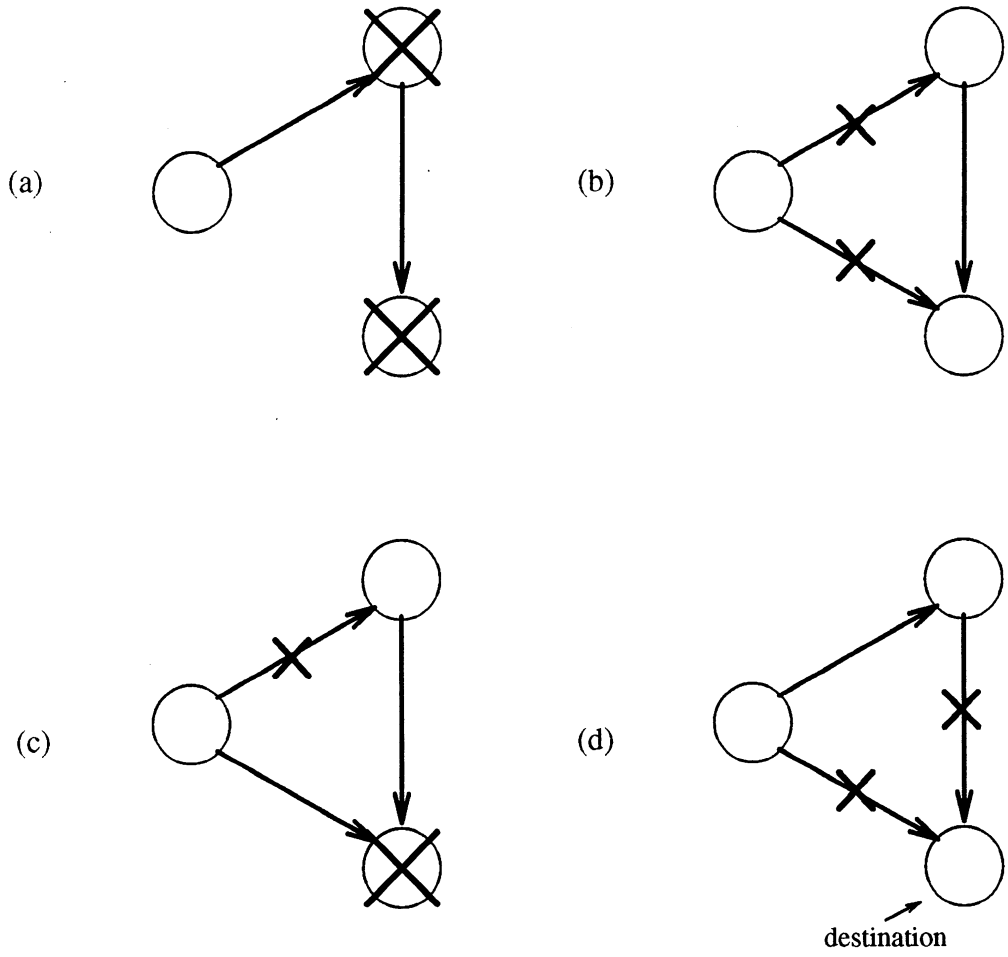

Figure 5.3: Four types of non-isolated component failures.

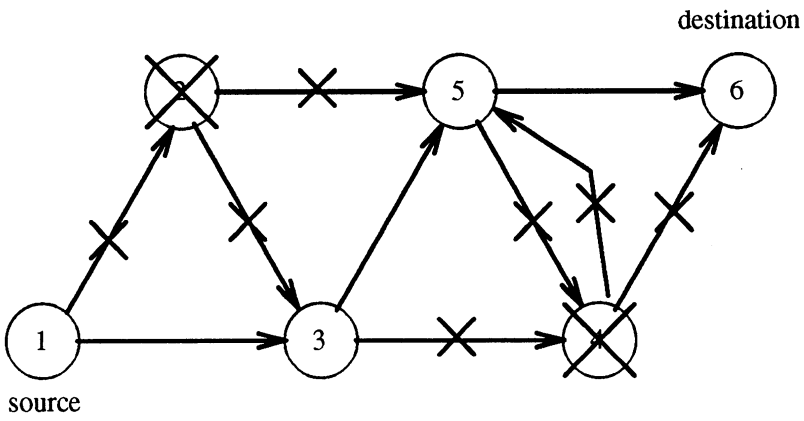

Figure 5.4: An IFI channel and one pattern of tolerable link/node failures.

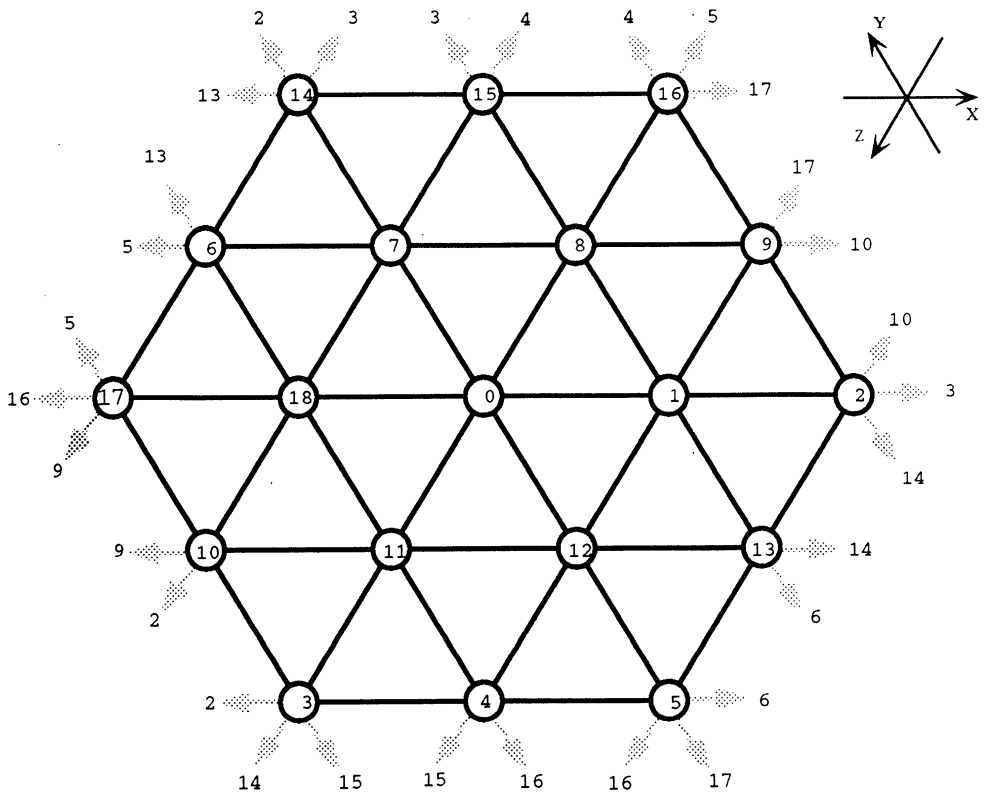

Figure 5.5: A wrapped hexagonal mesh of size 3.

[12]. HARTS has a *wrapped hexagonal mesh* interconnection network as shown in Figure 5.5 which can be defined as follows.

Definition 5.2 *Let* $[a]_b$ *denote* a *mod* b. *Then a wrapped hexagonal mesh of size* n *(or the number of nodes on each peripheral edge) is composed of* $N = 3n(n-1)+1$ *nodes, labeled from* 0 *to* $N-1$, *such that each node* s *has six neighbors* $[s+1]_N, [s+3n(n-1)]_N, [s+3n-2]_N, [s+3n^2-6n+3]_N, [s+3n^2-6n+2]_N$, *and* $[s+3n-1]_N$, *in the* $X, -X, Y, -Y, Z, -Z$ *directions, respectively.*

One important result obtained from the HARTS project is the routing algorithm in a wrapped hexagonal mesh network. It was proved in [25] that a wrapped hexagonal mesh is homogeneous. Consequently, any node can view itself as the center of the mesh. Let m_x, m_y, and m_z be, respectively, the number of hops (negative values mean the moves in negative directions) from the source node to the destination node along the X, Y, and Z directions on a shortest path. The following routing algorithm [25] determines the values of m_x, m_y, and m_z for the shortest paths from a source node s to a destination node d in a wrapped hexagonal mesh of size n:

Algorithm 5.5 (Routing in HARTS) .

Step 0. Set $m_x := 0$, $m_y := 0$, $m_z := 0$. Let $p = 3n^2 - 3n - 1$, $k = (d - s) \bmod p$, $r = (k - n) \operatorname{div} (3n - 2)$, $t = (k - n) \bmod (3n - 2)$.

Step 1. If $k < n$ then set $m_x := k$, stop. Else if $k > 3n^2 - 4n + 1$ then set $m_x = k - 3n^2 + 3n - 1$, stop. Else goto Step 2.

Step 2. If $t \leq n + r - 1$ then

- If $t \leq r$ then set $m_x := t - r$, $m_z := n - r - 1$, stop.
- If $t \geq n - 1$ then set $m_x := t - n + 1$, $m_y := r + 1 - n$, stop.
- Else set $m_y := r - t$, $m_z := n - t - 1$, stop.

else

- If $t \leq 2n - 2$ then set $m_x := t + 2 - 2n$, $m_y := r + 1$, stop.
- If $t \geq 2n + r - 1$ then set $m_x := t - 2n - r + 1$, $m_z := -r - 1$, stop.
- Else set $m_y := 2n + r - t - 1$, $m_z := 2n - t - 2$, stop.

We now discuss how real-time channels can be enhanced to be Isolated Failure Immune (IFI) in HARTS. The first step is to find an *IFI path*, which is defined as a subnetwork containing a directed path from the source to the destination in the presence of any isolated failures. Let $d_S(v_1, v_2)$ denote the minimum number of hops (i.e., distance) from node v_1 to node v_2 in a network S. The following theorem gives a sufficient condition for S to be an IFI path from a source node v_s to a destination node v_d.

Theorem 5.1 *A network S containing the source node v_s and the destination node v_d is an IFI path from v_s to v_d if*

C1 *Every node $v \in S, v \neq v_d$, has at least two outgoing links to two other nodes, say v_1 and v_2, such that $d_S(v_1, v_d) < d_S(v, d_d)$, $d_S(v_2, v_d) \leq d_S(v, v_d)$, and v_1, v_2 are adjacent,*

C2 *There is no loop in S whose nodes are all of the same distance $d > 1$ to the destination node v_d.*

Proof: From C1, every node $v \in S$ except the destination node has two outgoing links l_1 and l_2 which lead to a pair of adjacent nodes v_1 and v_2, respectively. Then, a packet will be blocked at node v only if (1) both l_1 and l_2 are disabled, or (2) both v_1 and v_2 are disabled, or (3) l_1 and v_2 are disabled, or (4) l_2 or v_1 are disabled. All these situations represent non-isolated failures. Thus, in the absence of non-isolated failures, a packet from the source node can always progress unless it has reached the destination. Further, C1 ensures a packet will not move away from the destination and C2 ensures that a packet will not move forever without reaching the destination node or circling in a loop in which each node is directly connected to v_d. Since v_d can not have more than one faulty incoming link, we conclude that a packet from the source node can always reach the destination node. □

From the above theorem, we see that each node in an IFI path needs only two outgoing links. We call one of them the *primary* link and the other the *secondary* link.

The primary link is the one which leads to a node closer to the destination. One can choose the primary link from the shortest path as determined by Algorithm 5.5. In case there exist multiple choices, i.e., more than one of the m_x, m_y, m_z are non-zero, we will use the following algorithm to select a primary link L.

Algorithm 5.6 (Selection of the primary link L) .
Let $abs(x)$ and $sign(x)$ denote the absolute value and the sign of x, respectively, and let X, $-X$, Y, $-Y$, Z, $-Z$ denote the outgoing links of a node along the six different directions. Then,

If $abs(m_x) > 1$ then set $L := sign(m_x)X$
else if $abs(m_y) > 1$ then set $L := sign(m_y)Y$
else if $abs(m_z) > 1$ then set $L := sign(m_z)Z$
else if $abs(m_x) = 1$ then set $L := sign(m_x)X$
else if $abs(m_y) = 1$ then set $L := sign(m_y)Y$
else if $abs(m_z) = 1$ then set $L := sign(m_z)Z$.

As will be clear later, the selection of the primary links in the way specified by Algorithm 5.6 will facilitate the determination of the secondary links and reduce the number of nodes/links of the resulting IFI channel.

To ensure that the secondary link does not lead to a node which is farther away from the destination, it must be either 60 degree above or 60 degree below the primary link.[2] We use the notation $L+1$ to denote the link which is 60 degree above L, and $L-1$ the one which is 60 degree below L. For example, if $L = X$, then $X + 1 = -Z$ and $X - 1 = -Y$.

Let $node[i]$ denote the ith node of an IFI path, and $node[i].p$ and $node[i].s$ denote the node's primary and secondary links, respectively. We propose the following algorithm to construct an IFI path from v_s to v_d.

Algorithm 5.7 (Construction of an IFI path) .

Step 1. Calculate m_x, m_y, m_z for all shortest paths from the source node v_s to the destination node v_d using Algorithm 5.5. Notice that at most two of them can be non-zero.

Step 2. Set $i := 1$ and $node[1] := v_s$. Set the initial rotating direction for the secondary link $R := 1$ if one of the following is true: (1) $abs(m_y) > abs(m_x) = 1$, (2) $abs(m_z) \geq abs(m_y) = 1$, (3) $abs(m_x) > 1$, $m_z \neq 0$, and (4) $abs(m_x) = abs(m_z) = 1$. Otherwise, set $R := -1$.

Step 3. Calculate the primary link $L(i)$ using Algorithm 5.6. If $i > 1$, $L(i) \neq L(i-1)$, and $node[i-1]$ is not adjacent to v_d, set $R := -R$.

Step 4. Set $node[i].p := L(i)$, $node(i).s := L(i) + R$, and set $node[i+1]$ to be the node which the secondary link of $node[i]$ leads to. Update m_x, m_y, m_z for $node[i+1]$.

Step 5. If $node[i+1] = node[i-1]$, then set $node[i+1] := v_d$ and stop. The destination node has been reached. Otherwise, set $i := i + 1$, $R := -R$, goto Step 3.

The correctness of Algorithm 5.7 is proved by the following theorem.

Theorem 5.2 *The subnetwork obtained from Algorithm 5.7 is an IFI path from v_s to v_d.*

Proof: We prove that the resulting subnetwork S satisfies C1 and C2 of Theorem 5.1.

For any $node[i] \neq v_d$ in S, let v_1 and v_2 be the two respective nodes which links $node[i].p$ and $node[i].s$ enter. From the algorithm, $node[i+1] = v_2$. Thus $v_2 \in S$. To show that v_1 is also in S, and v_1 and v_2 are adjacent, we first prove that there is a link in S from v_2 to v_1.

---

[2]Here "above" means counter-clockwise and "below" means clockwise.

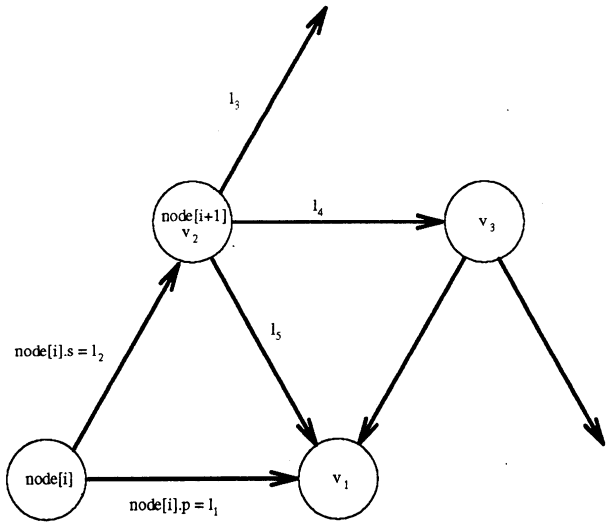

Figure 5.6: Proof of the adjacency of v_1 and v_2.

Since a secondary link will never lead to the destination node, $v_2 \neq v_d$. Thus, $node[i+1]$ always has two outgoing links $node[i+1].p$ and $node[i+1].s$ in S. Assume $node[i].s$ is 60 degree above $node[i].p$. As shown in Figure 5.6, from the direction of $node[i].p$ (which is on the shortest path from $node[i]$ to v_d), $node[i+1].p$ (i.e., the shortest path from $node[i+1]$ to v_d) has only three choices: l_3, l_4, l_5. We claim that $node[i+1].p$ can not take l_3 since otherwise, from Algorithm 5.6, $node[i].p$ would have taken l_2 instead of l_1. If $node[i+1].p = l_5$, the primary link of $node[i+1]$ is the link from v_2 to v_1. Otherwise, $node[i+1].p = l_4$. From Algorithm 5.7, $node[i+1].s$ should be 60 degree below $node[i+1].p$ since $node[i].p$ and $node[i+1].p$ have the same direction and $node[i]$ is not adjacent to v_d ($node[i+1].p$ would otherwise have taken l_5). Thus $node[i+1].s = l_5$ is the link from v_2 to v_1. Similarly, it can be proved that there is a link from v_2 to v_1 in S when $node[i].s$ is 60 degree below $node[i].p$.

We now prove that $v_1 \in S$. If $node[i+1].s = l_5$, then $v_1 = node[i+2] \in S$. Otherwise, from the above proof, $node[i+1].p = l_5$. If $v_1 = v_d$, from Algorithm 5.7, $node[i+1].s$ directs back to $node[i]$. Then, $v_1 = node[i+2] \in S$. Otherwise, as shown in Figure 5.6, $v_3 = node[i+2]$. Continuing this induction, we can conclude that either $v_1 \in S$, or the six neighbors of v_1 all have primary links directed to v_1. The latter case implies $v_1 = v_d$. Thus, $v_1 \in S$. Since there is a link in S from v_2 to v_1, v_1 and v_2 are adjacent in S.

Further, since $node[i].p$ is on the shortest path, $d_S(v_1, v_d) = d_S(node[i], v_d) - 1 < d_S(node[i], v_d)$. Since there exists a link in S from v_2 to v_1, $d_S(v_2, v_d) \leq d_S(v_1, v_d) + 1 = d_S(node[i], v_d)$. Thus C1 is proved.

We now prove that there does not exist any loop all of whose nodes are of a constant distance $d > 1$ to v_d by contradiction. First, notice that such a loop contains only secondary links since a primary link connects two nodes of different distances to v_d. Then, all the primary links of the nodes in the loop must lead to a common node v. This is from the fact proved above that either $node[i+1].p$ or $node[i+1].s$ must lead to node v which $node[i].p$ leads to. But $node[i+1].s$ can not lead to v since it must lead to a node of the same distance to v_d as that of $node[i]$. This is possible only if $v = v_d$, i.e., $d = 1$. Thus, C2 is proved. □

We make several remarks on Algorithm 5.7 as follows.

1. In Step 4, the address of $node[i+1]$ can be obtained from that of $node[i]$ using Definition 5.2, which gives the addresses of the six neighboring nodes of a node in six directions. The values of m_x, m_y, m_z for $node[i+1]$ can be updated directly with Algorithm 5.5 using the address of $node[i+1]$. But a simpler way of doing this is as follows. Let u be the direction of link $node[i].s$ and v, w be the remaining two directions. Let $s = 1$ if link $L(i) + R$ is at the positive direction of u and $s = -1$ otherwise. Then, if ($m_u = m_v = 0$ and $sm_w > 0$) or ($m_u = m_w = 0$ and $sm_v > 0$), update $m_v := m_v - s, m_w := m_w - s$. Otherwise, update $m_u := m_u - s$. The correctness of this algorithm can be verified by placing the destination node v_d at the center of the wrapped hexagonal mesh and checking the changes of m_x, m_y, m_y as one moves from $node[i]$ to $node[i+1]$ along link $node[i].s$.

2. In Step 2, the initial rotating direction R for the secondary link is chosen such that if $node[1]$ has two links both on shortest paths[3] to the destination nodes, $node[1].s$ will take one of them. In this way, the resulting IFI path needs less links and nodes than when doing otherwise. The way in which the primary link is chosen in Algorithm 5.6 also serves this purpose.

3. Since the primary links are always on the shortest path to the destination, they form a shortest path sinking tree to the destination. In other words, if a packet generated at any node in S is always forwarded using the primary links, it will take a minimum number of hops to the destination. This fact results in the following routing policy at each node: an arriving packet should be forwarded via the primary link whenever possible. The secondary link is used only if the primary link is down.

We now discuss how the IFI real-time channel can be established over an IFI path obtained from Algorithm 5.7. The procedures to establish an IFI real-time channel are composed of the following three steps.

Step 1. Calculate the message delay bound over each link of the channel.

Step 2. Calculate the end-to-end delay bound using the link delay bounds.

Step 3. If the end-to-end delay bound is not larger than the requested one, the channel can be established. Calculate the link delay bounds to be assigned to the channel. Otherwise, the channel establishment request is rejected.

Theorem 2.8 can be used for the calculation of the link delay bounds in Step 1. Let $node[i], i = 1, \cdots, k$ be the nodes of an IFI path obtained from Algorithm 5.7, where $node[1]$ is the source node and $node[k]$ is the destination node. Let $d[i].p$ and $d[i].s$ be the delay bounds over the primary and secondary links of $node[i]$, respectively. Then the end-to-end message delivery delay bound in Step 2 can be calculated using the following algorithm.

Algorithm 5.8 (Calculation of the message delivery delay bounds) .
The message delivery delay bound $d[i]$ from $node[i]$ to the destination node $node[k]$ for

[3]Note that there could be multiple shortest paths between a pair of nodes.

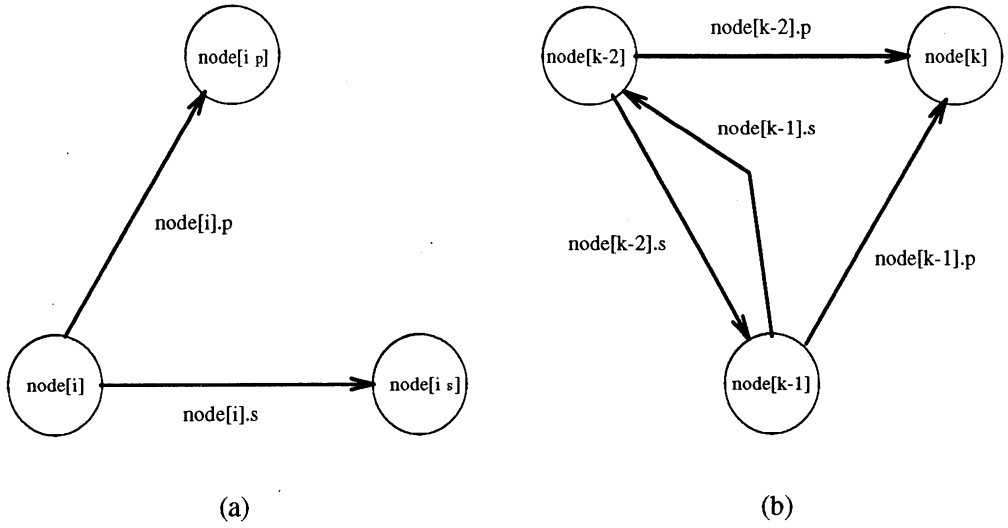

Figure 5.7: Calculation of $d[i]$'s.

a real–time channel with a maximum message transmission time C can be calculated as follows:

$$d[k-1] = \max\{d[k-1].p,\ d[k-1].s + d[k-2].p - \max\{0, (C - C_p)\}\},$$

$$d[k-2] = \max\{d[k-2].p,\ d[k-2].s + d[k-1].p - \max\{0, (C - C_p)\}\},$$

$$d[i] = \max\{d[i].p + d[i_p],\ d[i].s + d[i_s]\} - \max\{0, (C - C_p)\} \quad i = k-3, \cdots, 1.$$

where $node[i_p], node[i_s]$ are the nodes to which the primary and secondary links of $node[i]$ lead, respectively, and C_p is the packet transmission time.

The correctness of Algorithm 5.8 can be verified as follows. From the proof of Theorem 5.2, the connections between $node[k-2], node[k-1]$, and $node[k]$ are shown in Figure 5.7(b), from which the first two equations can be obtained using Theorem 3.1. For $1 \leq i \leq k-3$, $node[i]$ is connected to $node[i_p]$ and $node[i_s]$ in the way shown in Figure 5.7(a), which proves the remaining $k-3$ equations. Since i_p and i_s are always larger than i for $i \leq k-2$, the maximum delay bound from $node[i]$ to $node[k]$ can be obtained from the above equations.

If $d[1] \leq D$, the IFI real–time channel can be established, and we need to determine the link delay bounds to be assigned to the channel. As discussed in Section 5.2, the link delay bounds of the channel should be set as large as possible to reduce the channel's influence on the links' ability to establish more real–time channels in future. This can be done using the following algorithm.

Algorithm 5.9 (Assignment of link delay bounds) .

Step 1. In Algorithm 5.8, for $i = k-1, \cdots, 1$, record the link (i.e., the primary or secondary link) $l[i]$ on which the maximum is achieved for $d[i]$. Notice that there could be two links for $i = k-2$ or $i = k-1$.

Step 2. Record all the links traversed as one goes from $node[1]$ to $node[k]$ using only the links recorded in Step 1. This gives a critical path from the source to the destination which has the end–to–end delay bound $d[1]$ as calculated from Algorithm 5.8.

Step 3. Let N be the total number of links on the critical path. For each link ℓ_j on the critical path, set the channel's delay bound $d^j := d_j + (D - d[j])/N$, where d_j is the delay bound calculated using Theorem 2.8 for ℓ_j.

Step 4. Recalculate $d[i]$'s in Algorithm 5.8 with the link delay bounds on the critical path replaced by d^j's. The channel's delay bounds of the links not on the critical path can then be calculated as the differences of $d[i]$'s of the nodes they connect.

In summary, we have the following algorithm for the establishment of an IFI real-time channel.

Algorithm 5.10 (Establishment of an IFI real–time channel) .

Step 1. *Using Theorem 2.8, calculate the minimum message delay bounds* $d[i].p_{min}$ *and* $d[i].s_{min}$ *over the primary and secondary links of* $node[i], i = 1, \cdots, k-1$.

Step 2. *Calculate the end–to–end delay bound* $d[1]$ *from Algorithm 5.8.*

Step 3. *If* $d[1]$ *is larger than the user-requested end–to–end delay bound* D*, the channel request is rejected. Otherwise, the channel can be established with the link delay bounds calculated from Algorithm 5.9.*

We now give an example to demonstrate the above ideas. Figure 5.8 shows a portion of a hexagonal mesh. We want to establish an IFI real–time channel from node 1 to node 8 with channel parameters $(T, C, D) = (100, 5, 70)$. For simplicity, assume $C_p \geq C$.

We first construct an IFI path from node 1 to node 8 using Algorithm 5.7. For $i = 1$, $node[1]$ = node 1. $(m_x, m_y, m_z) = (2, 0, -2)$. The initial rotating direction for the secondary link $R = 1$ since $abs(m_x) > 1$ and $m_z \neq 0$. From Algorithm 5.6, the primary link is calculated to be $node[1].p = L(1) = X$, and the secondary link is $node[1].s = L(1) + 1 = -Z$.

Set the next node to one which link $-Z$ leads to, then $node[2]$ = node 2. Update m_x, m_y, m_z for $node[2]$ as follows. The direction of $-Z$ is Z, so $u = Z$, and $v = X, w = Y$. Also, $s = -1$. Since $m_w = 0$ and $sm_v = -2 < 0$, we only need to update $m_u := m_u - s = -2 + 1 = -1$. Thus, for node 2, $(m_x, m_y, m_z) = (2, 0, -1)$.

Repeating the above procedure, we get an IFI path as shown in Figure 5.8, where the primary links are denoted by solid arrows and the secondary links by dashed arrows. It is not difficult to see that a packet can be transmitted from node 1 to node 8 in the presence of any isolated failures. Also, all the primary links and the nodes form a shortest path sinking tree to the destination node.

We now establish an IFI real–time channel over the IFI path thus obtained by assigning delay bounds to the links using Algorithm 5.10. Suppose there is no other real–time traffic in the network. Then, for $i = 1, \cdots, 8$, $d[i].p_{min} = d[i].s_{min} = C = 5$. Using Algorithm 5.8, $d[i]$'s are calculated and shown near each node in Figure 5.8. The requested real–time channel can be established since $d[1] = 35 < D = 70$.

The critical path can be determined by recording the links over which the maximum is achieved in Algorithm 5.8, which is in this example the ones marked by "//" in

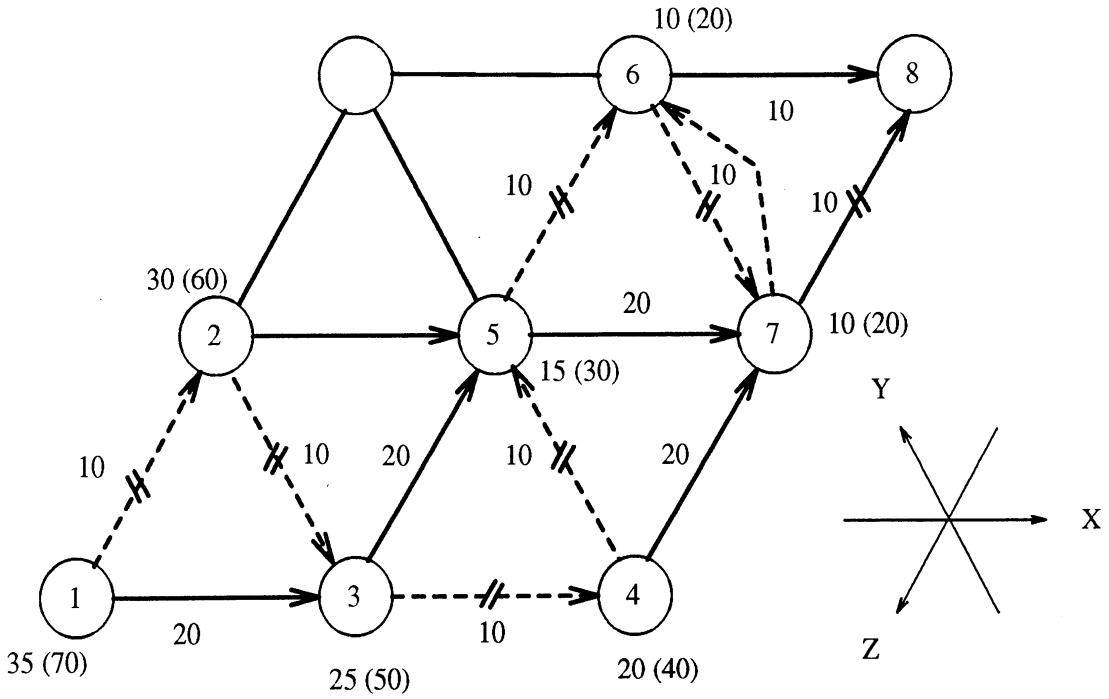

Figure 5.8: An IFI real–time channel from node 1 to node 8.

Figure 5.8. There are a total of $N = 7$ links on the critical path. The channel's delay bounds over the links of the critical path are thus $d^j = d_j + (D - d[1])/N = 5 + (70 - 35)/7 = 10$. The updated values of $d[i]$'s calculated from Algorithm 5.8 are shown in the parentheses near each node. Then, the channel's delay bounds on the other links can be calculated as the differences of $d[i]$'s of the nodes they connect, which are shown near each link in Figure 5.8.

5.4 Backup Channels

For real–time communication that can tolerate rare short–period break-downs, a less expensive way to increase the reliability of a real–time channel is to set up backup channels. This method is also applicable to cases where SFI circuits can not be established due to the poor connectivity of a network. The idea of the backup channels works as follows:

- Each real–time channel is composed of a *primary* channel and a number of *backup* channels. Under the normal circumstance, the primary channel is used for packet transmission while keeping the backup channels unused. In case the primary channel gets disabled by a component failure, one of the backup channels is promoted to the primary channel (thus taking over the transmission task).

- Both the primary and backup channels are established simultaneously using the procedures described in Chapter 3. In case a primary or backup channel cannot be established, the system is allowed to *tear down* some of the existing backup channels based on a scheme that will be described later.

Using backup channels, the long channel re-establishment overhead can be avoided when the primary channel is disabled. The cost of backup channels is minimal because:

- Backup channels are not used to transmit redundant real–time messages under the normal circumstance, thus they do not affect the transmission of other channels as well as non real–time traffic, and
- The number of real–time channels that a network can accommodate is not reduced since the backup channels can be removed whenever there is a shortage of network resources in establishing a new real–time channel.

As compared to the case of replicating real–time channels, there are two problems in implementing the idea of backup channels. First, there is a delay in switching to a backup channel when a fault occurs to the primary channel. The main source of this delay is associated with fault detection and channel switching. The real–time packets transmitted during this period could be lost. Second, there is no guarantee on the number of backup channels that each real–time channel can have. As more and more real–time channels are established in a network, the number of backup channels of a real–time channels may reduce and even become 0. In the rest of this section we will address how these two problems can be alleviated.

The fault detection time can be reduced by using an "acknowledgment channel" for each real–time channel. A channel fault can be detected quickly if the source node does not receive an ack in a certain period after transmitting a packet. Note that an acknowledgment channel usually costs far less than a real–time channel since it deals with a shorter packet size, a larger requested delivery delay (depending on the required fault detection time), and/or a longer packet inter-arrival time (several packets can be acknowledged at a time).

The channel switch time is the time needed to find a non faulty backup channel and promote it to the primary channel. If there is at least one non faulty backup channel, this process is quite fast. One can send multiple copies of a packet through all the backup channels of the now-disabled real–time channel, and choose the one which delivered the packet correctly. If, unfortunately, all the backup channels are faulty, there is no choice but to execute the time–consuming channel establishment procedure.

The second problem of using backup channels comes from the fact that a backup channel may be removed in order to accommodate future real–time channels. A real–time channel may have many backup channels when the network is lightly loaded (in the sense that there do not exist many real–time channels in the network). As more and more real–time channels are added, the number of backup channels may decrease or even become zero. Thus, the use of backup channels provide no guaranteed fault-tolerance for the primary channels. An important question is then how to manage the establishment and removal of backup channels such that the primary channels can be backed up as much as possible. To this end, the following three questions must be answered:

Q_1: For each real–time channel, how many backup channels should be established?

Q_2: How should the channels be routed?

Q_3: Which channel removal policy should be used?

A straightforward answer to Q_1 is that the more backup channels the better. A real–time channel is more likely to find a non faulty backup channel if it is backed up by

many channels. However, there is a limitation to the number of backup channels which can be established. Establishing too many backup channels will complicate the channel establishment procedure. Also, the system gains little by establishing channels which have a large number of common links and nodes, because a single component failure would bring all of them down. For this reason, we restrict that the primary and backup channels of a real–time channel run through *disjoint* paths, and the number of backup channels to be established is thus the maximum number of disjoint paths minus one.

As to the routing problem, we argued in Section 5.1 that the minimum–hop routing is preferable for real–time channels. The primary channel and backup channels can be set up sequentially by establishing one at a time, then removing all the intermediate nodes and links it passes through from consideration for setting up next channels, and establishing the next backup channel in the remaining network, and so on.

The third question consists of two parts: (i) which channels are allowed to be removed? and (ii) which of the removable channels should actually be removed?

As to part (i), establishment of a new channel should only be allowed to remove only those channels which are less important than itself. There are two ways to compare the relative importance of backup channels: (1) the backup channels of a critical real–time channel is more important than those of a less critical one, and (2) for two real–time channels of the same criticality, the backup channels of the one with more backup channels are less important than those of the other real–time channel. To this end, one may assign a criticality number C to each real–time channel and then assign a rank $R = f(C, k)$ to its k-th backup channel, where $f(C, k)$ is an increasing function of C and a decreasing function of k. One such choice is $f(C, k) = C - k$. So, we have a channel removal strategy which allows for tearing down only lower–rank backup channels. Since the establishment of a primary channel is always allowed to tear down backup channels, the rank of all primary channels is assigned to be ∞.

The answer to the second part of the question is that one should remove as few important backup channels as possible. In other words, if there is a choice between two backup channels, the one with lower rank should be removed. This can be done with the following algorithm.

Algorithm 5.11 (Channel Removal Strategy) .

Step 1. Establish the new channel without removing any existing backup channels. Terminate if successful. Otherwise, goto Step 2

Step 2. Remove all the backup channels on the route having lower ranks than the new one, except those with link delay bounds larger than t_m, the time that achieves the maximum of $d_n = \max\{d^t : t \in G\}$ in Theorem 2 (the removal of these channels will not change the link–delay bound of the new channel).

Step 3. Establish the new channel. If it is still not successful, the new channel cannot be established and those channels removed in Step 2 are restored. Otherwise, goto Step 4.

Step 4. Starting from the backup channels with the highest rank, re-establish the backup channels removed in Step 2. This step reduces the number of backup channels removed.

One possible application domain of the backup channels is in a dual-ring network (e.g., the FDDI) where both the SFI and IFI real-time channels can not be established. We give an example showing how backup channels can be established.

Consider a ring network with 5 stations connected by 5 duplex links (link i connects station i and $(i+1)$ mod 5). All real-time channels have the same minimum message inter-arrival time $T_i = 100$ and maximum message transmission time $C_i = 5$. Suppose three backup channels τ_{1b}, τ_{2b}, τ_{3b} have been established in the network with the following information on the source (src) and destination (dst) stations, user-requested end-to-end delay D_i, channel rank R_{ib}, and link-delay bound d_i^j. (To show the procedure of Algorithm 5.11 clearly, we do not consider the existing primary channels since the primary channels are not allowed to be torn down.)

τ_{1b}: src=3 dst=0 $D_1 = 15$ $R_{1b} = 1$ $d_{1b}^3 = 10$ $d_{1b}^4 = 5$

τ_{2b}: src=2 dst=0 $D_2 = 10$ $R_{2b} = 2$ $d_{2b}^1 = 5$ $d_{2b}^0 = 5$

τ_{3b}: src=3 dst=0 $D_3 = 30$ $R_{3b} = 1$ $d_{3b}^2 = 7$ $d_{3b}^1 = 12$ $d_{3b}^0 = 11$.

Now, we want to establish a new real-time channel τ_4 of criticality 4 with $src = 3$, $dst = 0$, $D_4 = 15$.

The primary channel of τ_4 takes the minimum-hop route $\ell_3 \rightarrow \ell_4$. Using Algorithm 2, one can establish the primary channel as:

τ_{4p}: src=3 dst=0 $D_4 = 15$ $R_{4p} = \infty$ $d_{4p}^3 = 5$ $d_{4p}^4 = 10$.

After removing ℓ_3 and ℓ_4 from the network, the network contains only one more connection between station 3 and station 0. So, τ_4 can have at most one backup channel τ_{4b}. Since the criticality of τ_4 is 4, the rank of its first backup channel is $R_{4b} = 4 - 1 = 3$. Setting up τ_{4b} on the route $\ell_2 \rightarrow \ell_1 \rightarrow \ell_0$ with Algorithm 2 results in rejecting the channel request. Hence, all other backup channels on the route with lower ranks lower than τ_4, i.e., τ_{2b} and τ_{3b}, are removed. After this removal, τ_{4b} can be successfully established as:

τ_{4b}: src=3 dst=0 $D_4 = 15$ $R_{4b} = 3$ $d_{4b}^2 = 5$ $d_{4b}^1 = 5$ $d_{4b}^0 = 5$.

The next step is to re-establish those backup channels removed, starting from the one with the highest rank. Using Algorithm 2, the establishment of τ_{2b} is rejected. So τ_{2b} must be removed and τ_{3b} is re-established as:

τ_{3b}: src=3 dst=0 $D_3 = 30$ $R_{3b} = 1$ $d_{3b}^2 = 10$ $d_{3b}^1 = 10$ $d_{3b}^0 = 10$.

Consequently, after establishing a new real-time channel τ_4, the network has the following four established channels:

τ_{1b}: src=3 dst=0 $D_1 = 15$ $R_{1b} = 1$ $d^3_{1b} = 10$ $d^4_{1b} = 5$

τ_{3b}: src=3 dst=0 $D_3 = 30$ $R_{3b} = 1$ $d^2_{3b} = 10$ $d^1_{3b} = 10$ $d^0_{3b} = 10$

τ_{4b}: src=3 dst=0 $D_4 = 15$ $R_{4b} = 3$ $d^2_{4b} = 5$ $d^1_{4b} = 5$ $d^0_{4b} = 5$

τ_{4p}: src=3 dst=0 $D_4 = 15$ $R_{4p} = \infty$ $d^3_{4p} = 5$ $d^4_{4p} = 10$.

5.5 Conclusion

We discussed how real-time fault-tolerant communication can be *simultaneously* achieved in computer networks. Unlike the traditional reliable communication protocols like the TCP which uses message retransmission, i.e., time redundancy, we explore spatial redundancy since time redundancy is usually not suitable for real-time communication.

CHAPTER 6

REAL–TIME CHANNELS OVER SHARED–MEDIUM LANS

6.1 Introduction

We have discussed how real–time channels can be established over point–to–point networks. In practice, however, an end system is usually connected to a shared–medium Local Area Network (LAN) and LANs are connected together by point–to–point networks. A message may have to go through one or more LANs to reach its destination. Thus, it is important to study how real–time channels can be established over shared–medium LANs.

A shared–medium LAN is usually composed of a common bus or ring which is shared by all the nodes in the network. Nodes' access to the bus or ring is controlled by a Medium Access Control (MAC) protocol. The MAC protocol can be contention–based or non contention–based.

A typical contention–based MAC protocol is the IEEE 802.3 (also called CSMA/CD) used for the Ethernet [27, 28] in which all nodes are connected together by a common bus. Each node is allowed to transmit its packets when it senses the idle bus. Due to the propagation delay, transmissions may collide when two nodes sense the idle bus and transmit their packets simultaneously. Thus a node needs to continue sensing the bus after the beginning of its packet transmission. When a collision is detected, the node stops transmission immediately, waits for a random amount of time, and re-transmits the packet when it senses the idle bus again. The Ethernet is the most widely used LAN today. However, due to its contention–based MAC protocol, an Ethernet can operate only at a low transmission speed (10 Mbps) and over a short distance (several hundred meters). Also, its performance (in terms of throughput and message delay) deteriorates quickly with the increase of traffic in the network. Since no deterministic medium access time is guaranteed, the Ethernet is not suitable for real–time communication.

Under non contention–based MAC protocols like the IEEE 802.4 token bus, IEEE 802.5 token ring, and the FDDI token ring [27, 28], the access to a shared–medium is controlled by a token circulating among nodes. Only the node which has the token is allowed to transmit packets, and the transmission time is controlled by a few internal timers of the node. In this way, the access to the medium can be regulated and transmission collisions can be avoided. Also, real–time services can be provided by bounding the worst–case access delay for each node. A ring network can operate at a high speed (e.g., 100 Mbps for the FDDI) over a long distance (up to a couple of hundred miles).

We will cover non contention–based MAC protocols only since real–time communication is inherently difficult to be realized with contention–based protocols. Among the token passing protocols, the FDDI is the newest, especially designed for high speed and

real–time networks. So we will use the FDDI to demonstrate our ideas. The results thus obtained are extensible to other token passing protocols like the IEEE 802.4 token bus and the IEEE 802.5 token ring.

This chapter is organized as follows. Section 6.2 reviews the MAC protocol of the FDDI and its relevant properties. Real–time channel establishment conditions over the FDDI is derived in Section 6.3, and protocols for using real–time channels over the FDDI are presented in Section 6.4. A modification to the MAC protocol of the FDDI, called the FDDI-M, is proposed in Section 6.5 which significantly improves the FDDI's capacity of supporting real–time traffic. The chapter concludes with Section 6.6.

6.2 Preliminaries

The Fiber Distributed Data Interface (FDDI) is a proposed ANSI standard for a 100 Mbps token ring network using a fiber–optic medium [29, 30]. During the past several years, a large number of FDDI networks have been installed and this number is still increasing rapidly. Thanks to its high transmission speed, the FDDI alleviates the bandwidth saturation problem of the current 10 Mbps Ethernet and the 4 Mbps or 16 Mbps IEEE 802.5 Token Rings. The FDDI's capacity of supporting synchronous traffic also makes it ideal for real–time applications.

Two types of traffic are supported in an FDDI network: synchronous and asynchronous. The network provides both a *bounded transmission delay* and a *guaranteed bandwidth* for synchronous traffic, while the asynchronous traffic is handled on a best–effort basis. These features are realized with the following MAC protocol.

Protocol 6.1 (MAC of the FDDI) .

P1: Suppose there are N nodes in a ring which are numbered from 0 to $N-1$. As part of an FDDI ring initialization process, each node declares a Target Token Rotation Time (TTRT). The smallest among them is selected as the ring's TTRT. Each node which supports synchronous traffic is then assigned a portion of the TTRT to transmit its synchronous messages. Let h_i, called the *high priority token holding time*, denote the portion of TTRT that node i is assigned to transmit its synchronous messages.

P2: Each node has two internal timers: the token–rotation–timer (TRT) and the token–holding–timer (THT). The TRT always counts up and a node's THT counts up only when the node is transmitting asynchronous packets. If a node's TRT reaches the TTRT before the token arrives at the node, the TRT is reset to 0 and the token is marked as late by incrementing the node's late count L_c by one. To initialize the timers at different nodes, no packets are allowed to be transmitted during the first token rotation after the ring initialization and L_c's are set to 0.

P3: Only the node that has the token is eligible to transmit packets. The packet transmission time is controlled by the node's timers, but an in-progress packet transmission will not be interrupted until its completion. When a node i receives the token, it does the following:

P3.1: If $L_c > 0$, set $L_c := L_c - 1$ and $THT := TTRT$. Otherwise, set $THT := TRT$ and $TRT := 0$.

P3.2: If node i has synchronous packets, it transmits them for a time period up to h_i or until all the synchronous packets are transmitted, whichever occurs first.

P3.3: If node i has asynchronous packets, it transmits them until the THT counts up to the TTRT or all of its asynchronous packets are transmitted, whichever occurs first.

P3.4: Node i passes the token to the next node $(i+1) \bmod N$. □

Let T_{ring} denote a ring's latency which is the time needed to circulate the token around the ring once without transmitting any packets, and T_p denote the time needed to transmit a maximum–size asynchronous packet. Then, the settings of the parameters of the FDDI MAC protocol must satisfy the following protocol constraint:

$$\sum_{i=0}^{N-1} h_i \leq TTRT - T_{ring} - T_p. \tag{6.1}$$

The physical meaning of the above inequality is that the summation of the assigned synchronous bandwidth over the nodes in the network should not exceed the effective ring bandwidth. Violation of this constraint would make the ring unstable and oscillate between "claiming" and "operational" [31]. Under this protocol constraint, a well–known fact about the FDDI is that the worst–case token rotation time is bounded by $2 \times TTRT$, and the average token rotation time is bounded by TTRT [32]. A more general result was obtained in a recent paper [33] as stated below.

Lemma 6.1 (Worst–case token rotation times) .
Under the protocol constraint (6.1) of the FDDI, the time elapsed between any n consecutive token's visits to a node i is bounded by $n \times TTRT - h_i$.

Once node i gets the token, it is given up to h_i units of time to transmit its synchronous packets. The following lemma gives a lower bound of time that node i is allowed to transmit its synchronous packets during a time period t [33].

Lemma 6.2 (Synchronous transmission time) .
Under the protocol constraint (6.1) of the FDDI, node i has at least $\lfloor t/TTRT - 1 \rfloor \cdot h_i$ *units of time to transmit its synchronous packets during a time period t. This lower bound is reached when* $\lceil t/TTRT - 1 \rceil \cdot TTRT - t \geq h_i$.

Lemmas 6.1 and 6.2 are the best published synchronous properties of the FDDI to date. In the next section, we will improve Lemma 6.2 and derive the conditions for establishing real–time channels over the FDDI.

6.3 Synchronous Bandwidth Allocation

A real–time channel τ passing through an FDDI network can be described by four parameters $\tau = (T, C, d, s)$, where s is the the source node of the channel in the network, T is the minimum logical inter-generation time of the channel's messages at s, C is the time needed to transmit a maximum–size message, and d is the requested message delivery delay bound over the ring. A real–time channel can be established over an FDDI network if the requested message delivery delay bound can be guaranteed by properly setting the parameters of the FDDI's MAC protocol.

The user adjustable parameters of the FDDI's MAC protocol are the TTRT and h_i's. TTRT is set at the ring initialization phase which determines the minimum message delay bound, $d_{min} = 2 \times TTRT$, that the network can support. Any channel request with a delay bound smaller than d_{min} will be rejected. With a given TTRT, a Synchronous Bandwidth Allocation (SBA) scheme determines the values of h_i's such that a given set of real–time channels can be established. An SBA scheme is said to be *feasible* with respect to a set of real–time channels if it can guarantee the requested transmission delay bounds of the channels. An SBA scheme is said to be *optimal* if it is always feasible whenever there exists a feasible SBA scheme.

Using Lemma 6.2, a normalized proportional SBA scheme was proposed in [33] which has the following features and/or problems.

1. The scheme can be used for real–time channels with $d_i = T_i$ only. In other words, it can establish only those real–time channels whose requested message delay bound d_i equal the message generation period T_i.

2. Using this scheme, an FDDI network is proven to be able to support any set of real–time channels with a total peak signal rate (i.e. $\sum_{i=1}^{n} C_i/T_i$) less than 33% of the ring bandwidth. This percentage was claimed to be the highest to date.

3. This scheme is *not* optimal. Thus a set of real–time channels which cannot be established with the normalized proportional SBA scheme could be established with some other scheme.

4. It is a *global* SBA scheme in that the allocation/deallocation of synchronous bandwidth to a node would require to change the synchronous bandwidths previously assigned to all other nodes.

The requirement of $d_i = T_i$ limits the type of applications that can be supported and the non-optimality of the scheme does not fully utilize the network's ability of accommodating synchronous traffic. Use of a global SBA scheme also complicates the synchronous bandwidth allocation protocol, making it difficult to implement.

As an improvement of the scheme in [33], Chen *et al.* proposed an optimal SBA scheme in a recent paper [34]. However, it still suffers the limitation of $d_i = T_i$ and is a global scheme. Besides, it uses an iterative algorithm for the calculation of the optimal bandwidths which may, in theory, need an infinite number of steps to converge.

In the following, we derive an SBA scheme which does not limit $d_i = T_i$ and is optimal when $d_i \leq T_i + TTRT$. The calculation of the optimal bandwidths can be done in just one step. Further, allocation/deallocation of a synchronous bandwidth to one node does not require to change the synchronous bandwidths assigned to other nodes, thus making the SBA scheme easy to implement.

Let $T(t)$ denote the time that a node in the worst–case is allowed to transmit its synchronous packets during a time period t. Lemma 6.2 gives a lower bound of $T(t)$ for node i. We extend Lemma 6.2 by calculating the exact value of $T(t)$ as follows.

Lemma 6.3 (Worst–case synchronous transmission time) .
Under the protocol constraint (6.1) of the FDDI, node i in the worst–case has

$$T(t) = \lfloor t/TTRT - 1 \rfloor h_i + \delta(t)$$

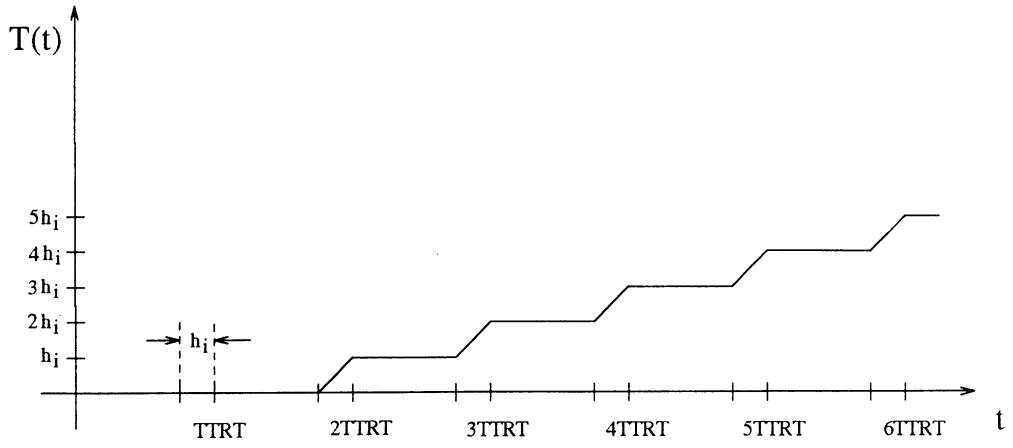

Figure 6.1: Worst–case synchronous transmission time $T(t)$.

units of time to transmit its synchronous packets during a time period t, where $\delta(t)$ is calculated as

$$\delta(t) = \begin{cases} 0 & if\ \lceil t/TTRT \rceil TTRT - t \geq h_i\ or\ t \leq TTRT \\ t - (\lceil t/TTRT \rceil TTRT - h_i) & otherwise. \end{cases}$$

Proof: $T(t)$ is plotted in Fig. 6.1. Its correctness can be seen from Lemma 6.1. In the worst–case, node i would first wait $2 \times TTRT - h_i$ units of time to get the token. Once it gets the token, it has h_i units of time to transmit its synchronous packets. This proves the correctness of $T(t)$ for $t \leq 2 \times TTRT$. From Lemma 6.1, the following worst–case token inter-arrival time at node i would be $TTRT$. This proves the correctness of $T(t)$ for $t > 2TTRT$. □

Suppose no two real–time channels have the same source node and the synchronous transmission time of a node is used for real–time channel messages only. Then from Lemma 6.3, we have the following necessary and sufficient condition for the establishment of a real–time channel over an FDDI network.

Theorem 6.1 (Channel establishment conditions over the FDDI) .
A real–time channel $\tau = (T, C, d, s)$ can be established over an FDDI network under the protocol constraint (6.1) if and only if

$$\forall t \geq 0, \quad \lceil (t-d)/T \rceil^+ C \leq T(t), \tag{6.2}$$

where $T(t)$ is calculated from Lemma 6.3 with $i = s$.

Proof of the necessary condition: Suppose node s does not have any message of channel τ at time $t = 0$. Then, $\forall t > 0$, a necessary condition for no messages to miss their deadlines in $[0, t]$ is that the amount of time, $\tau(t)$, needed to transmit all those messages generated during $[0, t]$ by channel τ with deadlines $\leq t$ is not greater than $T(t)$, the time that node s in the worst–case is allowed to transmit its synchronous packets. Since the minimal message inter–generation time of channel τ is T, there are at most $\lceil (t-d)/T \rceil^+$ messages generated by channel τ during $[0, t]$ with deadlines $\leq t$, which need at most $\lceil (t-d)/T \rceil^+ C$

units of time to transmit. Thus, the maximum value of $\tau(t)$ is $\lceil (t-D)/T \rceil^+ C$. This proves the necessary condition.

Proof of the sufficient condition: We prove this by contradiction. Suppose a message misses its deadline at time t_1, meaning that at least one message with deadline $\leq t_1$ has *not* been transmitted by t_1. Then there must exist $t' < t_1$ such that during the time period $[t', t_1]$, node i uses *all* its allowable synchronous transmission time for channel τ's packets. Let t_0 be the smallest such t', then there are no messages with deadlines $\leq t_1$ queued at the link at time t_0^-. Thus, in the time period $[t_0, t_1]$, node i uses all its synchronous transmission time transmitting only those packets of τ which are generated during $[t_0, t_1]$ with deadlines $\leq t_1$. Based on the same reasoning as the proof of the necessary condition, the maximum amount of time needed to transmit these messages is $\tau(t_1 - t_0) = \sum_{i=1}^{n} \lceil (t_1 - t_0 - D_i)/T_i \rceil^+ C_i$. Since one message misses its deadline at t_1, this $\tau(t_1 - t_0)$ must be larger than $T(t_1 - t_0)$, that is,

$$\lceil (t_1 - t_0 - D)/T \rceil^+ C > T(t_1 - t_0).$$

By letting $t = t_1 - t_0$, the above inequality contradicts the condition that $\forall t \geq 0$, $\lceil (t - D)/T \rceil^+ C \leq T(t)$. □

Since the left-hand side of Eq. (6.2) changes only at points $t = d + kT$ with value $\lceil ((d + kT) - d)/T \rceil^+ C = (k+1)C$, we have the following corollary from Theorem 6.1.

Corollary 6.1 .
A real-time channel $\tau = (T, C, d, s)$ can be established over an FDDI ring under the protocol constraint (6.1) if and only if h_s is set such that

$$h_s \geq \alpha_s C \tag{6.3}$$

where $\alpha_s = \max\{(k+1)/(T(d+kT)/h_s) : \ k = 0, 1, \cdots\}$.

Then, we have the following SBA scheme for the establishment of a real-time channel.

Algorithm 6.1 (Channel establishment over the FDDI) .
Suppose $n-1$ real-time channels $\tau_i = (T_i, C_i, d_i, s_i)$, $i = 1, \cdots, n-1$ have already been established over an FDDI ring. Then a new channel $\tau_n = (T_n, C_n, d_n, s_n)$ can be established with the following steps.

Step 1: *Calculate α_n from Corollary 6.1 and $h_{s_n} = \alpha_n C_n$.*

Step 2: *If the protocol constraint (6.1) is satisfied, set the high-priority token holding time of s_n to be h_{s_n} and establish channel τ_n. Otherwise, the channel establishment request is rejected.* □

Algorithm 6.1 gives an optimal SBA scheme since it uses the sufficient and necessary channel establishment condition of Corollary 6.1. In other words, if a real-time channel cannot be established with Algorithm 6.1, so cannot with any other SBA schemes. However, one problem with Algorithm 6.1 is the calculation of α_n. The definition of α_n in Corollary 6.1 is not given in closed-form. Thus, we need the following theorem for the calculation of α_n.

Theorem 6.2 (Calculation of α_n) .
Let $x = d_n/TTRT - 1$ and $y = T_n/TTRT$. Then,

$$\alpha_n = \begin{cases} 1/\lfloor x \rfloor & if\ y \geq \lfloor x \rfloor \geq 1 \\ 1/y & if\ y \leq 1\ and\ x \geq 2 \end{cases}$$

and

$$\alpha_n \leq \begin{cases} 1 + (2-x)/y & if\ y \leq 1\ and\ 1 \leq x < 2 \\ 1/\lfloor y \rfloor & if\ 1 < y < \lfloor x \rfloor. \end{cases}$$

Proof: By setting $k := 0$ and $k := \infty$ in the definition of α_n, we have $\alpha_n \geq 1/\lfloor x \rfloor$ and $\alpha_n \geq 1/y$, respectively.

From Lemma 6.3, $T(t) = \lfloor t/TTRT - 1 \rfloor h_n + \delta(t) \geq \lfloor t/TTRT - 1 \rfloor h_n$. Thus, $\alpha_n \leq \max\{(k+1)/\lfloor (d_n + kT_n)/TTRT - 1 \rfloor : k = 0, 1, \cdots\} = \max\{(k+1)/\lfloor x + ky \rfloor : k = 0, 1, \cdots\}$. Let $\alpha_n(k) = (k+1)/\lfloor x + ky \rfloor$.

For $y \geq \lfloor x \rfloor \geq 1$, $\alpha_n(k) = (k+1)/\lfloor x + ky \rfloor \leq (k+1)/((k+1)\lfloor x \rfloor) = 1/\lfloor x \rfloor$. Thus, $\alpha_n \leq 1/\lfloor x \rfloor$. Together with the fact that $\alpha_n \geq 1/\lfloor x \rfloor$, we have $\alpha_n = 1/\lfloor x \rfloor$.

For $y \leq 1$ and $x \geq 2$, $\alpha_n(k) = (k+1)/\lfloor x + ky \rfloor \leq (k+1)/(x - 1 + ky) \leq (k+1)/(1+ky) \leq (k+1)/((1+k)y) = 1/y$. Thus, $\alpha_n \leq 1/y$. Together with the fact that $\alpha_n \geq 1/y$, we have $\alpha_n = 1/y$.

For $y \leq 1$ and $1 \leq x < 2$, $\alpha_n(k) \leq \min\{f_1(k), f_2(k)\}$, where $f_1(k) = k + 1$ and $f_2(k) = (k+1)/(x - 1 + ky)$. It is easy to see that $f_1(k)$ is an increasing function and $f_2(k)$ a decreasing function of k. At $k = (2-x)/y$, $f_1(k) = f_2(k) = 1 + (2-x)/y$. Thus, $\alpha_n \leq 1 + (2-x)/y$.

For $1 < y < \lfloor x \rfloor$, $\alpha_n(k) \leq (k+1)/(\lfloor x \rfloor + k\lfloor y \rfloor) \leq (k+1)/((k+1)\lfloor y \rfloor) = 1/\lfloor y \rfloor$. □

The values of α_n in different regions of the x–y plane are plotted in Fig. 6.2. We need not consider the case of $x \leq 1$ since it means $d_n < 2 \times TTRT = d_{min}$ and the channel cannot be established. In most cases, the inequality $d_n \leq T_n + TTRT$ is satisfied, meaning that $y \geq \lfloor x \rfloor$. So, the exact value $\alpha_n = 1/\lfloor x \rfloor = 1/\lfloor d_n/TTRT - 1 \rfloor$ can be obtained and an optimal SBA scheme is realized via Algorithm 6.1. For regions on the x–y plane where the exact value of α_n cannot be obtained, one can use an upper bound of α_n instead, with little loss of accuracy, because the difference between the upper bound and the actual value of α_n is always smaller than 1.

Comparing with the SBA schemes of [33] and [34], Algorithm 6.1 has the following advantages.

Generality: The SBA schemes of [33, 34] can establish real-time channels with $d_i = T_i$ only, while Algorithm 6.1 can establish channels of arbitrary parameters, i.e., $d_i \leq T_i$ or $d_i > T_i$. This extension is very important in practice since for many applications, especially those in real-time control/monitoring systems, the required delay bound d_i is usually smaller than the message inter-generation period T_i. Real-time channels with $d_i > T_i$ are also useful for multimedia applications. Thus restricting $d_i = T_i$ would greatly limit a network's ability and effectiveness of supporting real-time communications.

Optimality: The SBA scheme of [33] is not optimal, even under the restrictive assumption $d_i = T_i$. Thus a real-time channel establishment request may be rejected even if it can be established using another scheme. The SBA scheme of [34] is optimal

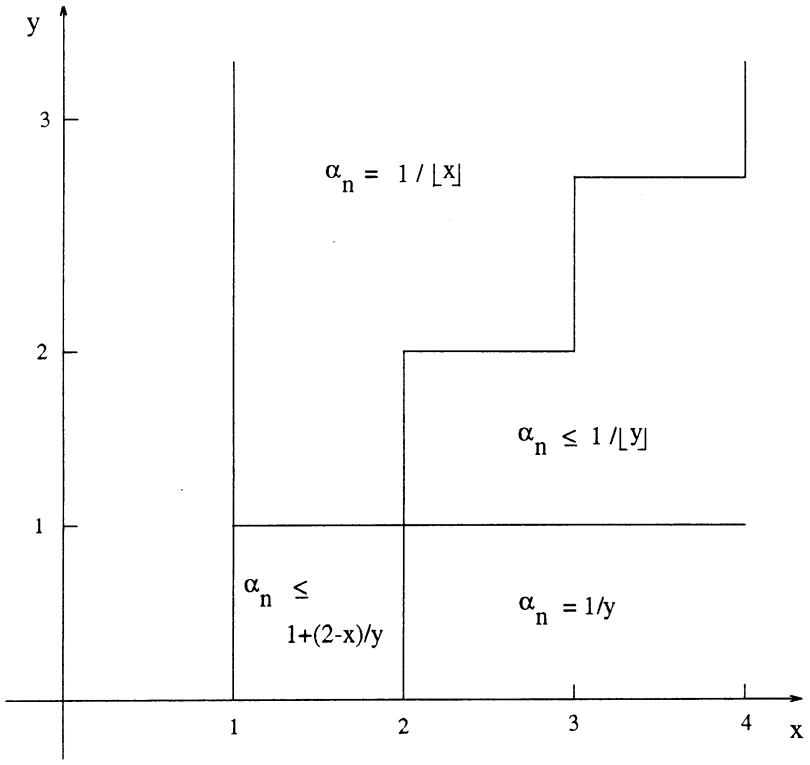

Figure 6.2: Calculation of α_n.

under the restrictive assumption of $d_i = T_i$ and requires complex computations. By contrast, Algorithm 6.1 is optimal for $d_i \leq T_i + TTRT$ (which subsumes the special case $d_i = T_i$ of [33, 34]) as well as for some other cases when the exact value of α_n can be calculated (see Algorithm 6.2), because it is based on the necessary and sufficient conditions of Theorem 6.1, and the computation of the optimal bandwidths is simple and straightforward. Rejection of a channel establishment request by Algorithm 6.1 means the violation of the necessary conditions, implying that the channel cannot be established with any other scheme.

Simplicity: The SBA schemes of [33, 34] are *global* schemes in the sense that the addition/removal of a channel or change of the parameters of a channel would require adjustment of the high-priority token holding times of all nodes in the network. This requires a complex SBA protocol. By contrast, Algorithm 6.1 needs only *local* parameter adjustment, thereby making it far easier to implement than those in [33, 34].

We now investigate an FDDI network's capacity of supporting real–time communication. Given a set of real–time channels $\tau_i = (T_i, C_i, d_i, s_i)$, $i = 1, \cdots, n$, its utilization is defined as $U = \sum_{i=1}^{n} C_i/T_i$. U_w is said to be the *worst–case achievable utilization* of a network if the network can accommodate every set of real–time channels with utilization $\leq U_w$, and for any $U > U_w$, there exists a set of real–time channels with utilization U which can not be accommodated by the network [33]. The following theorem gives the worst–case achievable utilization of an FDDI network using Algorithm 6.1.

Theorem 6.3 (Worst–case achievable utilization of FDDI) .
Let $\lambda = \min\{d_i/T_i : i = 1, \cdots, n\}$. *Suppose* $d_i \leq TTRT + T_i$ *and ignore* T_{ring} *and* T_p, *then Algorithms 6.1 and 6.2 have the worst–case achievable utilization* $U_w = \lambda/3$.

Proof: From Algorithm 6.1, a set of real-time channels $\tau_i = (T_i, C_i, d_i, s_i)$, $i = 1, \cdots, n$, can be established if and only if $\sum_{i=1}^{n} h_{s_i} = \sum_{i=1}^{n} \alpha_i C_i = \sum_{i=1}^{n} (C_i/T_i)(\alpha_i T_i) \leq TTRT - T_{ring} - T_p$. Ignoring T_{ring} and T_p, we have the channel establishment condition

$$\sum_{i=1}^{n} (C_i/T_i)(\alpha_i T_i/TTRT) \leq 1.$$

From Algorithm 6.2, when $d_i \leq TTRT + T_i$, $\alpha_i T_i/TTRT = (T_i/d_i)(d_i/TTRT)/\lfloor d_i/TTRT - 1 \rfloor$. Since $d_i/TTRT \geq 2$, $(d_i/TTRT)/\lfloor d_i/TTRT - 1 \rfloor < 3$. Thus $\alpha_i T_i/TTRT < 3/\lambda$. We conclude that the channel establishment condition is always satisfied if the channel utilization $U = \sum_{i=1}^{n} C_i/T_i \leq \lambda/3 = U_w$. For any $U > U_w$, it is also easy to construct a set of real-time channels with utilization U such that the channel establishment condition is violated. □

Under the condition that $d_i \leq T_i + TTRT$, Algorithms 6.1 and 6.2 give an optimal SBA scheme, thus the U_w given in Theorem 6.3 is the worst-case achievable utilization of an FDDI network. In other words, no other SBA scheme can guarantee the establishment of a set of real-time channels with utilization $> U_w = \lambda/3$. In [33], Agrawal *et al.* proved that their normalized proportional SBA scheme has a worst-case achievable utilization of 33% when $d_i = T_i$. Thus their scheme, albeit not optimal, reaches the highest worst-case achievable utilization.

Since the increase of d_i will not affect the establishment of a real-time channel, an FDDI network guarantees the successful establishment of any set of real-time channels with utilization lower than 33%, no matter how large d_i's are. U_w decreases linearly with the decrease of λ when $\lambda < 1$. This means that the smaller the requested delay bounds, the more difficult to establish the channels.

Another useful metric to measure the ability of a network to establish real-time channels is the *best-case achievable utilization* U_b, which is defined as the highest utilization of a set of real-time channels that the network can support. Any set of real-time channels with a utilization larger than U_b is guaranteed to be rejected. Clearly, U_b should reach 1 for an ideal network. Unfortunately, this is not the case for the FDDI as shown in the following theorem.

Theorem 6.4 (Best-case achievable utilization of FDDI) .
Suppose $d_i \leq T_i + TTRT$ and ignore T_{ring} and T_p, the best-case achievable utilization of an FDDI network is $U_b = \max\{\lfloor d_i/TTRT - 1 \rfloor/(T_i/TTRT) : i = 1, \cdots, n\}$.

The proof of Theorem 6.4 is similar to that of Theorem 6.3. To see how restrictive U_b is, let's consider a special case when all channels are identical with $d_i = T_i = T, i = 1, \cdots, n$ and $TTRT = T/2$. Then from Theorem 6.4, $U_b = 1/2$, which means that an FDDI network can use at most one half of its transmission bandwidth to set up real-time channels in this case.

From Theorems 6.3 and 6.4, we see that the FDDI's MAC protocol is not very efficient in supporting real-time communication. We will propose a modified protocol, called FDDI-M, to improve the FDDI's real-time performance in Section 6.5.

6.4 Real-time Channels over the FDDI

Similar to that for point-to-point networks, usage of real-time channels over an FDDI network needs two protocols: a message transmission protocol and a channel estab-

lishment protocol.

The message transmission protocol deals with message packetization and transmission scheduling. Message packetization is necessary since (1) the message size may exceed the maximum frame size of the FDDI, and (2) the high priority token holding time assigned to a node may not be large enough to transmit one whole message. The results of Section 6.3 were obtained under the assumption that a message can be divided into packets of any size and there are no packetizing overheads. Clearly, this is not very realistic in practice. In this section, we assume that all messages are divided into packets of the same size P, and each packet has an overhead of O bits. Then, a message of M bits will be split into $k = \lceil M/P \rceil$ packets with a total length of $M + kO$ bits. In the real–time channel model, this packetization overhead can be taken into account by letting C denote the time needed to transmit a maximum–size message *after* it is split into packets.

Another effect of using a fixed packet size is that the high priority token holding time may expire in the middle of a packet transmission. From the MAC protocol of the FDDI, an in-progress packet transmission will not be interrupted until its completion. Thus, node i may use up to $h_i + T_p$ units of time to transmit synchronous packets, where T_p is the time needed to transmit a packet of size P. This phenomenon is called *packet overrun.* Due to the packet overruns, the protocol constraint needs to be modified as follows:

$$\sum_{i=0}^{N-1} (h_i + T_{p_i}) \leq TTRT - T_{ring} - T_p. \tag{6.4}$$

where $T_{p_i} = T_p$ if $h_i > 0$ and $T_{p_i} = 0$ otherwise.

Transmission scheduling is necessary when more than one real–time channel originate from the same source node. The high priority token holding time of the node must be used to transmit urgent packets first. As in the point–to–point networks, we use the deadline scheduling policy again, i.e., each packet is assigned a deadline which equals the logical arrival time of the message to which the packet belong plus the requested delay bound over the FDDI network. When a node gets the token, it transmits the packet with the earliest deadline first. With the deadline scheduling policy, the required high priority token holding time for a node from which more than one real–time channel originates can be calculated as follows: if a channel τ_1 requires $h_i = t_1$ and a channel τ_2 requires $h_i = t_2$, then setting $h_i = t_1 \dot{+} t_2$ will satisfy the requirements of both channels.

In summary, we have the following message transmission protocol.

Protocol 6.2 (Message transmission over FDDI) .

P1: On the generation (arrival) of a message, use Protocol 3.1 (Protocol 3.2) to divide the message into packets and assign a deadline to each packet.

P2: On arrival of the token, transmit packets in order of their deadlines until the high priority token holding time expires or all of the synchronous packets are transmitted, whichever occurs first.

The channel establishment protocol implements Algorithm 6.1 to establish real–time channels. To establish a real–time channel, the source node calculates the required high priority token holding time locally and check if the protocol constraint (6.4) is satisfied. To do the latter, a global variable $T_S = \sum_{i=1}^{N-1}(h_i + T_{p_i})$ is needed. Since the token visits each station regularly, it is convenient to let the token hold this variable. This idea leads to the following channel establishment protocol.

Protocol 6.3 (Real–time channel establishment over FDDI) .
A real–time channel $\tau = (T, C, d, s)$ can be established by the source node s with the following steps:

Step 1: If $d < 2 \times TTRT$, the channel request is rejected. Otherwise, goto Step 2.

Step 2: Calculate the required high priority token holding time increment $\delta h_s = \max\{T_p, \alpha C\}$, where α is calculated from Algorithm 6.2. Let $\delta_p = T_p$ if τ is the only real–time channel originating from node s, and $\delta_p = 0$ otherwise. Wait until the token arrives and goto Step 3.

Step 3: Read T_S from the token. If $T_S + \delta h_s + \delta_P \leq TTRT - T_{ring} - T_p$, goto Step 4. Otherwise, the channel request is rejected.

Step 4: Update $T_S := T_S + \delta h_s + \delta_P$, and $h_s := h_s + \delta h_s$. The requested channel is established. Node s can start transmitting channel τ's message using Protocol 6.2.

A real–time channel can be removed by updating $h_s := h_s - \delta h_s$ at the source node, and $T_S := T_S - \delta h_s - \delta_P$ in the token.

6.5 A New MAC Protocol: FDDI-M

The token passing scheme used for the MAC protocol of the FDDI was proposed in 1982 [35]. Since then, a large number of papers have been published uncovering various properties of the protocol. However, few have discussed the rational and possible improvements of the protocol itself. Actually, from Theorems 6.3 and 6.4 of Section 6.3, we see that the MAC protocol of an FDDI network is not very efficient for real–time communication. For example, in some cases, the network can use at most 50% of its transmission bandwidth to support real–time traffic. In this section, we examine the reasons for this and propose a modification to the FDDI, called the FDDI-M, which can significantly improve an FDDI network's ability of supporting real–time communication.

The main reason for the FDDI's deficiency in supporting real–time communication is that the target token rotation time TTRT cannot always control the actual token rotation time below TTRT. As proved in [30], the worst–case token rotation time could be twice as large as TTRT. Thus, to guarantee an access delay bound, say d, the $TTRT$ must be set to no larger than $d/2$ instead of d. Due to the protocol constraint of the FDDI, a reduction of the TTRT results in the reduction of the synchronous transmission time h_i's, thus reducing the network's capacity of supporting synchronous traffic.

To see the above reasoning more clearly, let us calculate the maximum number of identical real–time channels with message inter-arrival time T, maximum message transmission time C, and requested message delay bound $d = T$ that can be established over an FDDI network. Since the worst–case token rotation time could be as large as $2 \times TTRT$, TTRT must be set to no larger than $T/2$ in order for a node to get the token at least once every T units of time. Suppose the TTRT is set to be $T/2$ (As discussed in [29], TTRT should not be set too small in order to achieve a high network efficiency), the high–priority token holding time of the node then must be set to no smaller than C in order to give the node enough time to transmit at least one message after getting the token. Thus, from the protocol constraint of the FDDI, at most $n = (TTRT - T_{ring} - T_p)/h_i < T/(2C)$ channels can be established. Since each channel can carry at most $b = C/T \times 100$ Mbps of traffic, an

FDDI network can carry no more than $n \times b = 50$ Mbps of real–time traffic, which is only one half of the network's capacity.

If the worst–case token rotation time can be controlled under the target token rotation time TTRT, TTRT then can be set to T instead of $T/2$, and twice as many of real–time channels (or equivalently, real–time traffic) can be established in an FDDI network. To realize this, notice that the token rotation time is composed of three components of time: for the transmission of synchronous packets, asynchronous packets, and the ring latency T_{ring}. During one token's rotation, the time used for the transmission of synchronous packets is bounded by $T_S = \sum_{i=0}^{N-1} h_i$ (ignoring the synchronous packet overruns). So, if we can control the time used for the transmission of asynchronous packets not to exceed $T_A = TTRT - T_S - T_{ring}$, the maximum token rotation time will never exceed the TTRT. This can be implemented by making the following two modifications to the standard FDDI's MAC protocol.

M1: Use a modified token rotation time $TTRT_m := TTRT - T_S - T_p$ instead of the TTRT.

M2: Stop the counting of a node's token rotation timer (TRT) when a synchronous packet is being transmitted/forwarded by the node.

With these modifications, the token holding timer (THT) at each node ensures the time used by the node for transmitting asynchronous packets plus the time used for transmitting asynchronous packets during the last token's rotation not to exceed $TTRT_m + T_p - T_{ring} = TTRT - T_S - T_{ring} = T_A$, as long as $T_S = \sum_{i=0}^{N-1} h_i \leq TTRT - T_{ring}$. Thus the maximum token rotation time will never exceed $T_A + T_S + T_{ring} = TTRT$.

The above modifications are easy to implement. One only needs to modify the timer circuits such that the counting of a node's TRT is stopped when the node is transmitting/forwarding a synchronous packet and resumed when the node starts transmitting/forwarding an asynchronous packet or the token. Notice that a packet's type is identified by the first bit of the Frame Control (FC) field in the packet's header [30], thus no need to change the packet format. The use of the late counter L_c can also be eliminated since the token will never be late under the modified token passing protocol. Making the above modifications to the standard FDDI MAC protocol leads to a new protocol, called the FDDI-M, as follows.

Protocol 6.4 (FDDI-M) .

P1: As part of an FDDI-M ring initialization process, each node declares a TTRT which equals the requested transmission delay bound of its synchronous messages. The smallest among all TTRTs is selected as the ring's TTRT. Each node which supports synchronous traffic is then assigned a portion of TTRT to transmit its synchronous packets. Let $h_i \geq 0$ denote the portion that node i is assigned to transmit its synchronous messages. Update $TTRT := TTRT - \sum_{i=0}^{N-1} h_i - T_p$, where T_p is the time needed to transmit a maximum–size packet.

P2: Each node has two timers: the token-rotation-timer (TRT) and the token-holding-timer (THT). A node's THT counts up when the node is transmitting an asynchronous packet and the TRT counts up when the the node is not transmitting/forwarding a synchronous packet. To initialize the timers at different nodes, no packets are allowed to be transmitted during the first token rotation after the ring initialization.

P3: Only the node which has the token is eligible to transmit packets. The packet transmission time is controlled by the timers, but an in-progress packet transmission will not be interrupted. When node i receives the token, it does the following.

P3.1: $THT := TRT$ and $TRT := 0$.

P3.2: If node i has synchronous packets, it transmits them for a time period up to h_i or until all the synchronous packets are transmitted, whichever occurs first.

P3.3: If node i has asynchronous packets, it transmits them until the THT counts up to the TTRT or all the asynchronous packets are transmitted, whichever occurs first.

P3.4: Node i passes the token to the next node $(i+1) \bmod N$. □

Similar to Lemma 6.3, we have the following lemma for the calculation of the time that a node in the worst–case is allowed to transmit its synchronous packets during a time period t.

Lemma 6.4 (Worst–case synchronous transmission time of FDDI-M) .
Under the protocol constraint (6.1), node i in the worst–case has

$$T(t) = \lfloor t/TTRT \rfloor h_i + \delta(t)$$

units of time to transmit its synchronous packets during a time period t, where $\delta(t)$ is calculated as

$$\delta(t) = \begin{cases} 0 & if\ \lceil t/TTRT \rceil TTRT - t \geq h_i \\ t - (\lceil t/TTRT \rceil TTRT - h_i) & otherwise. \end{cases}$$

With the above lemma, a real–time channel can be established over an FDDI-M network as follows.

Algorithm 6.2 (Channel establishment over FDDI-M) .
Suppose $n-1$ real–time channels $\tau_i = (T_i, C_i, d_i, s_i)$, $i = 1, \cdots, n-1$ have been established over an FDDI-M ring. Establishment of a new channel $\tau_n = (T_n, C_n, d_n, s_n)$ can be done with the following steps:

Step 1: If $d_n < TTRT$, the channel establishment request is rejected. Otherwise, goto Step 2.

Step 2: Calculate α_n from Algorithm 6.3 and $h_{s_n} = \alpha_n C_n$. If the protocol constraint (6.1) is satisfied, set the high priority token holding time of s_n to be h_{s_n}. Channel τ_n can be established. Otherwise, the channel establishment request is rejected. □

Algorithm 6.3 (Calculation of α_n for FDDI-M) .
Let $x = d_n/TTRT$ and $y = T_n/TTRT$. Then,

$$\alpha_n = \begin{cases} 1/\lfloor x \rfloor & if\ y \geq \lfloor x \rfloor \geq 1 \\ 1/y & if\ y \leq 1\ and\ x \geq 2 \end{cases}$$

and

$$\alpha_n \leq \begin{cases} 1 + (2-x)/y & if\ y \leq 1\ and\ 1 \leq x < 2 \\ 1/\lfloor y \rfloor & if\ 1 < y < \lfloor x \rfloor. \end{cases}$$

The only difference between Algorithm 6.2 and Algorithm 6.3 is the definition of x.

The worst–case achievable utilization of the FDDI-M can be calculated as follows.

Theorem 6.5 (Worst–case achievable utilization of FDDI-M) .
Let $\lambda = \min\{d_i/T_i : i = 1, \cdots, n\}$. Suppose $d_i \leq TTRT + T_i$ and ignore T_{ring} and T_p, then Algorithms 6.2 and 6.3 have the worst–case achievable utilization $U_w = \lambda/2$.

One can see that the FDDI-M can achieve significantly higher U_w than the FDDI. For $\lambda \geq 1$, an FDDI-M network guarantees the successful establishment of any set of real–time channels with utilization lower than $U_w = 50\%$, up from 33% with the FDDI.

The best–case achievable utilizations of the FDDI-M can be calculated as follows.

Theorem 6.6 (Best–case achievable utilization of FDDI-M) .
Suppose $d_i \leq T_i + TTRT$ and ignoring T_{ring} and T_p, the best–case achievable utilization of an FDDI-M network is $U_b = \max\{\lfloor d_i/TTRT \rfloor/(T_i/TTRT) : i = 1, \cdots, n\}$.

Since $\lfloor d_i/TTRT \rfloor > \lfloor d_i/TTRT - 1 \rfloor$, FDDI-M can always achieve a higher U_b than FDDI. For the special case considered for Theorem 6.4 where $d_i = T_i = T$ for all channels, TTRT can be set to T with the FDDI-M. Thus, $U_b = 1$ from Theorem 6.6 meaning that the full transmission capacity of the network can be used to support real–time traffic. In other words, the network's capacity of supporting real–time channels is doubled in this case.

The proofs of the above results are similar to that for the FDDI and are thus omitted here. With the enhanced capacity of supporting synchronous traffic, a natural question is then what is the FDDI-M's ability of supporting asynchronous traffic? Surprisingly, the modified protocol also improves a ring's ability of supporting asynchronous traffic.

To see this clearly, notice that the 100 Mbps transmission bandwidth of an FDDI ring is always partitioned into three parts: those used for the transmission of synchronous messages, asynchronous messages, and token passing. During one rotation of the token, the time used for token passing is a constant, T_{ring}. Thus, the faster a token rotates, the more ring bandwidth is wasted for passing the token. In other words, for a given synchronous throughput, the larger an average token rotation time that a protocol can achieve, the more asynchronous traffic it can support. Suppose the synchronous message generation period is T, then from the discussion in Section 1, the largest average token rotation time that a standard FDDI protocol can achieve is $TTRT = T/2$. Using the FDDI-M, on the other hand, the nodes always have $TTRT_m - T_{ring} - T_p$ units of time to transmit asynchronous messages. When a ring is heavily loaded with asynchronous traffic, the average token rotation time is at least $TTRT_m = T - T_S - T_p$. Ignoring T_p and since T_S should always be controlled under $TTRT = T/2$ in an FDDI network, the FDDI-M can always achieve a higher asynchronous traffic throughput than the standard FDDI.

Another salient feature of an FDDI network is its fairness in transmitting asynchronous messages [36]. If a number of nodes have a large amount of asynchronous traffic to transmit, then all nodes will achieve approximately an identical average asynchronous throughput. FDDI-M preserves this feature. This can be seen by observing that for every node with many asynchronous messages to transmit, the average time it uses to transmit asynchronous messages during each token's visit is $TTRT_m$ minus the average time used to transmit asynchronous messages during one token's rotation (ignoring packet overruns). Thus, on average, each node has approximately the same amount of time to transmit its asynchronous messages at each token's visit.

In summary, the proposed FDDI-M protocol has the following advantages:

1. It significantly improves the FDDI's capacity of supporting synchronous traffic. This is of great significance as more and more applications require real–time services of computer networks.

2. It preserves all the good features of an FDDI network, e.g., the high efficiency and fairness in supporting asynchronous traffic. Thus, comparing to the FDDI, the FDDI-M gains a lot and loses nothing.

3. Its implementation requires only minimal modification to the existing FDDI. Basically, only a few minor changes to the MAC layer timer circuits are sufficient. No modifications to the FDDI's frame format and upper layer software are needed. Thus, a large number of existing FDDI networks can be easily upgraded to the FDDI-M with virtually no additional cost and transparently to users.

6.6 Conclusion

This section addresses the problem of establishing of real–time channels over the FDDI networks. A modified protocol, called the FDDI-M, is also proposed which significantly improves the FDDI's ability of supporting real–time communication.

A complete scheme for the establishment of real–time channels in computer networks has been presented while showing the feasibility and advantages of using real–time channels to improve quality of service of the contemporary computer networks. We will discuss the application of real–time channels for multimedia networking in the next chapter.

CHAPTER 7

MULTIMEDIA APPLICATION

7.1 Introduction

This chapter discusses the application of real–time communication for multimedia networking. With the dramatic increases in link bandwidth and node processing power at inexpensive costs, real-time transmission of multimedia information like digital motion video over computer networks has now become a reality. Extensive work has been reported on the development of video processor, multimedia workstations, and network support for this new application [37, 38, 39, 40, 41].

Traditionally, video signals are transmitted over dedicated circuits only, e.g., the CATV. Clearly, this kind of service lacks flexibility (mainly for broadcasting or fixed point–to–point communication) and is not easily adaptable to computer networks for which packet switching is commonly used. To remedy this inflexibility, a hybrid transmission protocol, called the FDDI-II, has been proposed to add time–division circuit–switched transmission to the existing packet–switched FDDI networks [29, 42]. In an FDDI-II network, users can set up circuits with bandwidth of multiples of 64 Kbps up to 98.304 Mbps. In this way, continuous media communication like digital voice and video transmissions can be readily accommodated.

However, the problem with the fixed–bandwidth circuits is the inefficient use of link bandwidth [37]. The bit–rate of digital video signals, after compressed, is highly time–varying, with the peak rate several times higher than the average rate. To ensure the smooth transmission of video frames, the circuit bandwidth must be set significantly higher than the average signal bandwidth, thus wasting a great deal of circuit bandwidth. Since the bandwidth of video signals is usually very high (from several Mbps to several hundred Mbps), this kind of waste of circuit bandwidth is highly undesirable, even in high–speed networks.

By statistically multiplexing packets on transmission links, packet switched transmission is believed to be able to support variable bit–rate transmissions and make more efficient use of link bandwidth [37]. However, due to the contention delays at intermediate nodes, packet–switched transmission does not guarantee timely delivery of packets. This causes a serious problem in transmitting real–time motion video where each video frame is required to be delivered in a timely manner.

The concept of real–time channels studied in this thesis is a perfect way to solve the above problem. A real–time channel has the advantages of both circuit–switched and packet–switched transmissions. Specifically, it guarantees the end–to–end message transmission delay bound as a dedicated circuit, and at the same time, reserve the high transmission efficiency of packet–switched computer networks.

In this chapter, we investigate the transmission of compressed motion video over computer networks with various protocols. Section 7.2 compares the real–time channels with both circuit–switched and packet–switched transmissions. Section 7.3 compares the FDDI-M with the FDDI and the FDDI-II in local area ring networks. The chapter concludes with section 7.4.

7.2 Real–time Channels

As discussed before, the transmission of compressed digital motion video requires both *variable* link bandwidth and *timely* delivery of video frames. Using fixed–bandwidth circuits wastes link bandwidth and the packet–switched transmission cannot guarantee the timely delivery of video frames. In this section, we verify these claims with simulations and show that real–time channels outperform both circuit and packet switched transmission with respect to the utilization of link bandwidth and the timeliness of frame delivery.

7.2.1 Simulation Models

The network used for simulation is a 100 Mbps ring with 20 nodes. When video transmission is introduced, a 100 Mbps ring could easily get congested and thus become the bottleneck of the system.

The objective of our simulation is to evaluate and compare the transmission delays of video frames using circuit switching, packet switching, and real–time channels. The video data used here are obtained from a sequence of CNN headline news, stored on a laser disk. The size of each frame is 512 × 512 black and white pixels. The number of bits in each frame, after compressed with JPEG [43], is plotted in Fig. 7.1. At a 30 frames/second transmission rate, the video sequence needs an average 10.5 Mbps, and a peak 15.3 Mbps transmission bandwidth.

The frame–transmission delays using a fixed–bandwidth circuit are easy to calculate. Let τ_i be the time when the ith frame is generated at the source node, and t_i be the time when the last bit of the ith frame has been transmitted. Ignoring the propagation delays, the delay experienced by the ith frame is $d_i = t_i - \tau_i$. Starting from time 0 and assuming a 30 frames/second transmission rate, τ_i's and t_i's can be recursively calculated as follows:

$$\tau_i = i/30, \quad i = 1, 2, ...$$

$$t_i = \max\{t_{i-1}, \tau_i\} + S_i/B, \quad i = 1, 2, ...$$

$$t_0 = 0$$

where B is the bandwidth of the circuit, and S_i is the number of bits in the ith frame.

The frame-transmission delays using packet switching or real-time channels cannot be derived analytically. We wrote a detailed simulator (an 1100-line C program) to emulate the real transmission process and observe the delay each individual frame experienced. The simulation model of the node architecture which supports real-time channels is shown in Fig. 7.2.

In our simulation, each video frame is divided into packets of 10 Kbits each. The packet deadlines are calculated as described in Chapter 3. The packets received from the up–ring or generated at the local host are first put in the scheduler buffer *sbuffer*, and then

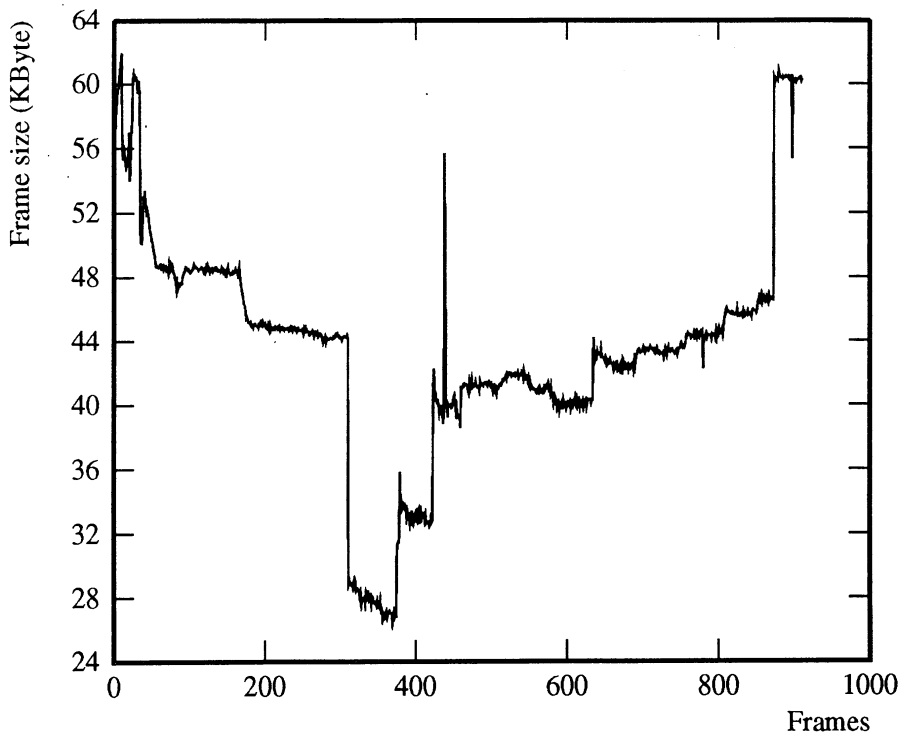

Figure 7.1: Video frame sizes.

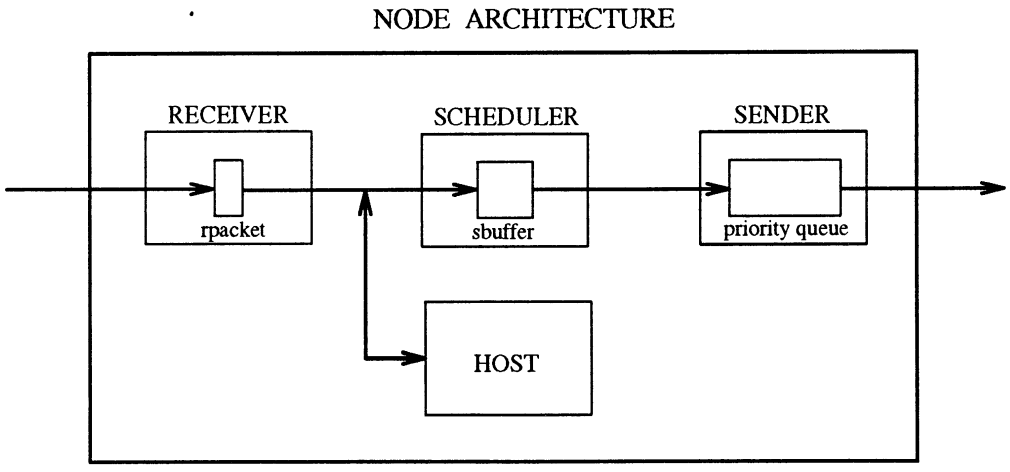

Figure 7.2: The simulation model of node architecture for real-time channels.

are inserted in the *priority queue* according to their deadlines. When the sender completes the transmission of a packet, it fetches and transmits the next packet at the head of the priority queue. See Chapter 4 for a detailed account of scheduling process.

The same simulator is used for packet switching with one change: packets from *sbuffer* are queued at the tail of the priority queue. This reflects the fact that packet transmission is usually scheduled on a First-In-First-Out (FIFO) basis in an ordinary packet switching node.

7.2.2 Real-time channels vs. circuit switching

Link efficiency: As stated earlier, the main problem with the circuit switching is the waste of transmission bandwidth. There is a tradeoff between the circuit bandwidth and video-frame transmission delays. Reserving too little circuit bandwidth introduces large frame delays, while reserving too much bandwidth wastes transmission capacity since no one else is allowed to use the reserved bandwidth. Our first experiment is thus to investigate the relation between the circuit bandwidth and the maximum frame delay.

We use the FDDI-II in our simulation since it is a well-defined protocol and supports circuit-switched transmissions in packet-switched ring networks. The simulation results are plotted in Fig. 7.3, where the x-axis represents the circuit bandwidth normalized by the average signal bandwidth (10.5 Mbps), and the y-axis represents the maximum frame delays normalized by the frame period (33.3 ms). From Fig. 7.3, one can see that the maximum frame delays are very sensitive to the circuit bandwidth allocated. For the video sequence used in our simulation (Fig. 7.1), the maximum frame delay is as large as 32 frame periods, or approximately 1 second, if the circuit bandwidth is set to be the average signal bandwidth. To make the maximum frame delay smaller than one frame period, the circuit bandwidth has to be at least 1.5 times the signal bandwidth, wasting 50% of transmission capacity.

With real-time channels, on the other hand, no network capacity will be wasted. The average bandwidth used by a real-time channel is always equal to the average bandwidth of the signals transmitted over the channel. When a real-time channel is idle, the full link bandwidth can be used to trasfer other traffic.

Another disadvantage of the FDDI-II is that an established circuit always runs through every node in the network regardless of the distance between the source and destination. This aggravates the problem of wasting circuit bandwidth since it requires the reservation of some links' bandwidth even if they are not used for communicating packets between the source and destination. If we define the network bandwidth as the total number of links in the network times the bandwidth of each link, then the network bandwidth available for other traffic after establishing N FDDI-II circuits with bandwidth B in our 100 Mbps 20-node ring would be $2000 - 20NB$ Mbps which is plotted in Fig. 7.4 with $B = 15.75$ Mbps. With real-time channels, on the other hand, only those links which connect the source and destination nodes are used. In a 20-node ring network, the average number of links needed to connect a pair of source and destination nodes is 10. Thus, the network bandwidth available after establishing N real-time channels is $2000 - 10NB_s$ Mbps (as plotted in Fig. 7.4) on the average, where $B_s = 10.5$ Mbps is the video signal bandwidth. From Fig. 7.4, we see that real-time channels use significantly less network bandwidth than the FDDI-II.

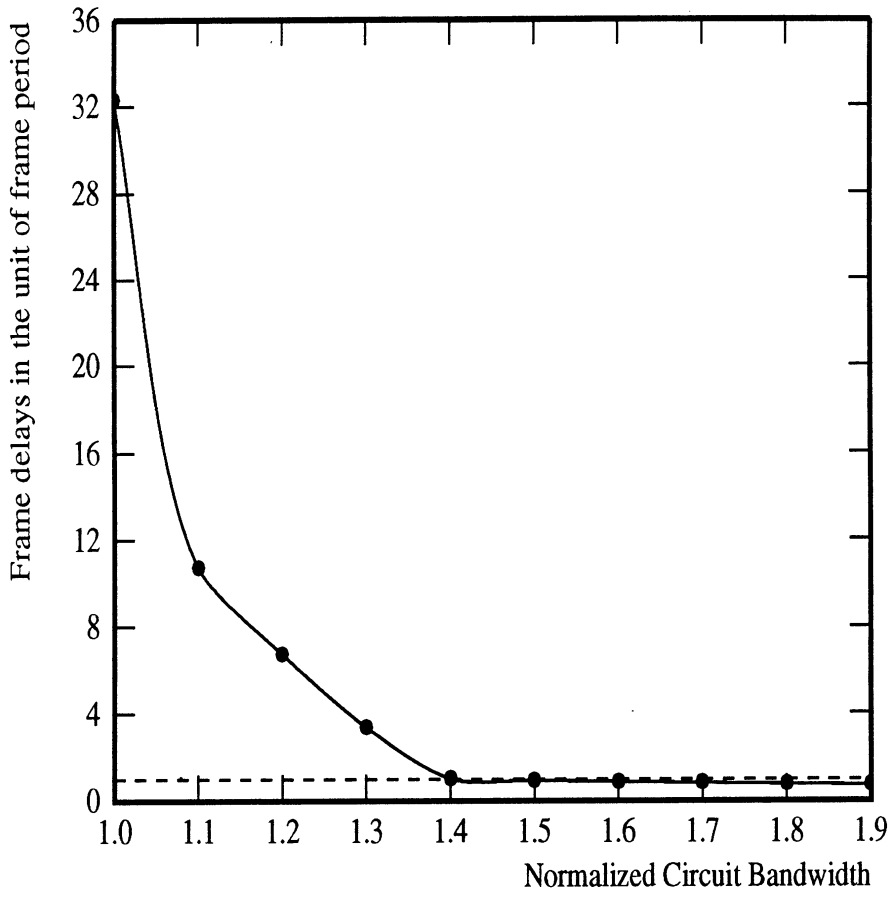

Figure 7.3: Maximum video frame delays with circuit switching.

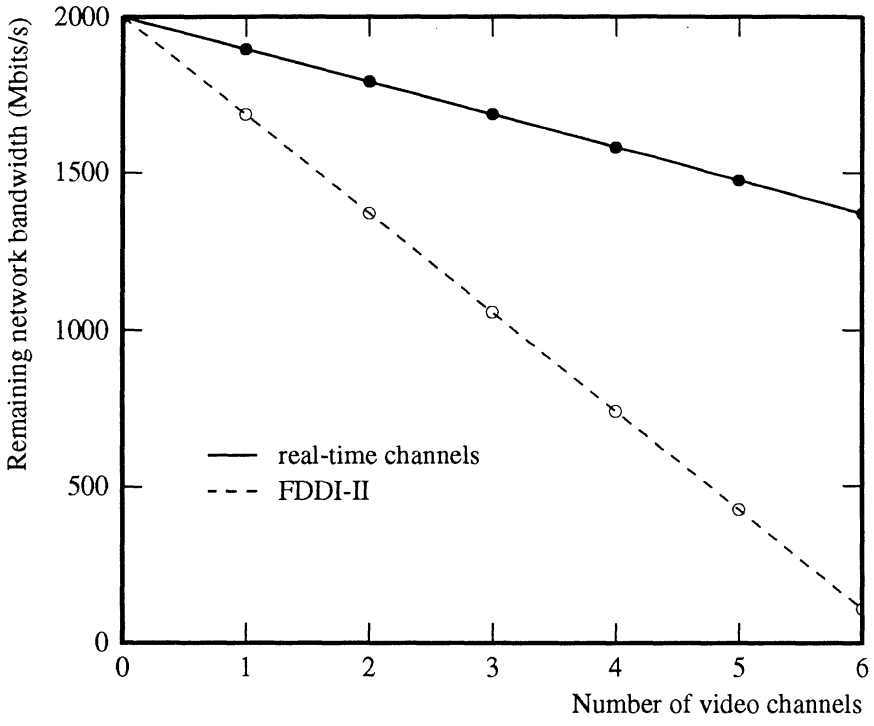

Figure 7.4: The remaining network bandwidth after establishing video channels.

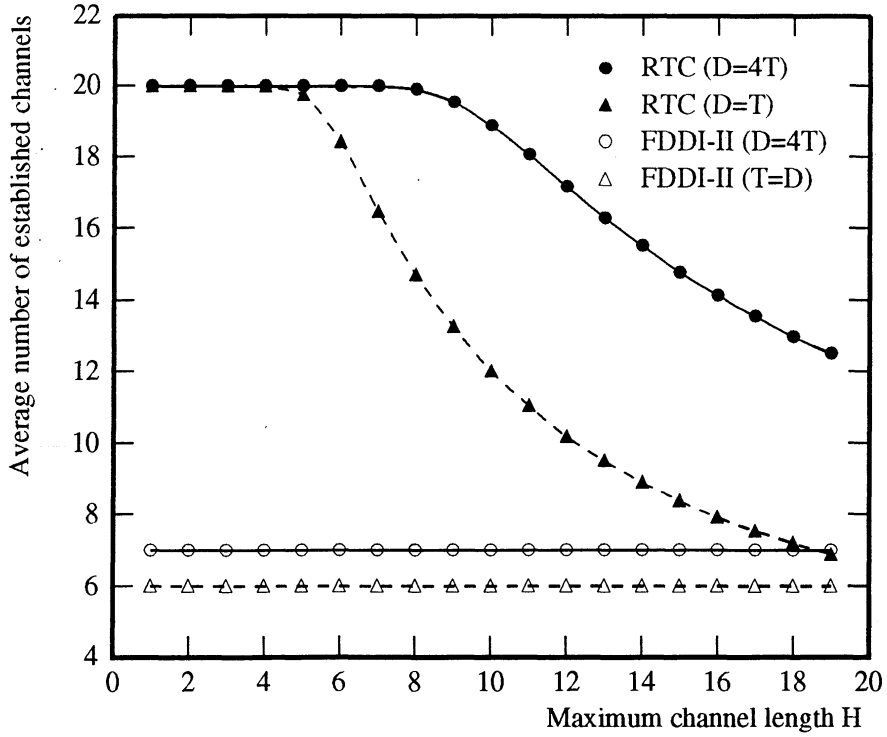

Figure 7.5: Average number of successfully established video channels with real–time channels and the FDDI-II. The lengths of real–time channels are uniformly distributed in $[1, H]$.

Ability to accommodate video channels: We now compare the number of video channels that each protocol can support. As stated earlier, the FDDI-II allows users to set up circuits with bandwidth of multiples of 64 Kbps up to 98.034 Mbps in a single–ring network. From Fig. 7.3 we see that to make video frame–transmission delay smaller than one frame period, the circuit bandwidth should be at least 1.5×10.5 Mbps = 15.75 Mbps. With the total available bandwidth of 98.034 Mbps, at most $98.034/15.75 = 6$ video channels can be established.

The number of video channels that can be established with real–time channels depends on the locations of the source and destination nodes. We assume that each host node wants to set up a video channel to a destination node which is h links away, where h is a random integer variable uniformly distributed over $[1, H]$, and $1 \leq H \leq 19$ is an integer parameter (H cannot be larger than 19 in a 20-node ring). For each H, we use Protocol 3.3 to establish real–time channels for each node. The parameters of the real–time channels are set to: minimum message inter-arrival time $T = 33$ ms (30 frames/second), maximum message transmission time $C = 5$ ms (500 Kbits maximum message size), requested end–to–end delay bound $D = 33$ ms (one frame period), and the packet transmission time $C_p = 0.1$ ms (10 Kbits packet size). Each experiment was repeated 10000 times. Out of 20 requested channels, the average numbers of establishable channels for different values of H are plotted in Fig. 7.5 (the dashed curve).

From this figure, we see that real–time channels outperform the FDDI-II by far

when the channel length is short. The number of establishable channels decreases as the channel length increases.

Another interesting feature is that one can significantly increase the ability of the real-time channels to accommodate video channels by relaxing end-to-end delay bounds. For example, if each destination node has a buffer capacity to store 4 video frames, then the end-to-end frame delivery delay bound can be relaxed to be four frame periods, i.e., $D = 4T = 120$ ms. From Fig. 7.3, we see that with the FDDI-II, the circuit bandwidth can be reduced to be 1.3 times the average signal bandwidth, i.e., 1.3×10.5 Mbps = 13.65 Mbps. Thus, 98.034/13.65 = 7 video channels can be established in an FDDI-II network. The number of establishable video channels with real-time channels for $D = 4T$ is also plotted in Fig. 7.5 (the solid curve), which shows a significant improvement over FDDI-II.

7.2.3 Real-time channels vs. packet switching

Ordinary packet switching exhibits two problems when used for real-time communication. First, its FIFO or Round-Robin scheduling policy treats all packets equally. Urgent messages do not receive the requisite transmission priority, and thus could easily miss their deadlines. Second, no efficient traffic control scheme is used to prevent network from congestion. When the network gets congested, real-time messages will get delayed or lost as well as the non real-time messages.

With the real-time channel protocol, urgency of a message is represented by the deadlines of its packets. The deadline scheduling of packet transmissions gives priority to the most urgent packets. The amount of real-time traffic over each link is controlled by the channel establishment algorithm that guarantees the timely delivery of all real-time messages of any established channel. The existence of non real-time traffic in the network has virtually no effect on real-time channels since non real-time packets always have lower transmission priority than real-time packets. When the network is congested with excessive non real-time traffic, only non real-time messages will get delayed or lost.

We verify the above claims with simulations. 19 video channels were established in the 20-node ring network described in Section 3.1. Channel 0 covers the whole ring, i.e., with the source node 0 and the destination node 19. Channel i, $i = 1, \cdots, 18$, has the source node i and the destination node $i + (3 - (i - 1) \bmod 3)$. All 19 channels transmit the video frames displayed in Fig. 7.1. The maximum frame transmission delay is required to be smaller than one frame period.

A certain amount of non real-time traffic is generated with random source and destination nodes. The amount of traffic is measured by the percentage of the network bandwidth needed to transmit them. For example, 10% of non real-time traffic in a 100 Mbps ring means that each link will carry an average of 10 Mbps of non real-time packets. Recall that each video channel carries signals of an average bandwidth of 10.5 Mbps, and up to 4 video channels are established over some links with the source and destination nodes selection described above. Thus the network will get congested if the non real-time traffic over one link exceeds 100 Mbps $- 4 \times 10.5$ Mbps = 58 Mbps, or the total non real-time traffic exceeds 58% of the network bandwidth.

The maximum and average frame delays of channel 0 are plotted in Fig. 7.6. Fig. 7.7 plots the distribution of frame delays. One can see that real-time channels are virtually independent of the amount of non real-time traffic. The maximum and average delays keep well below the requested one frame period, even when the network is nearly congested.

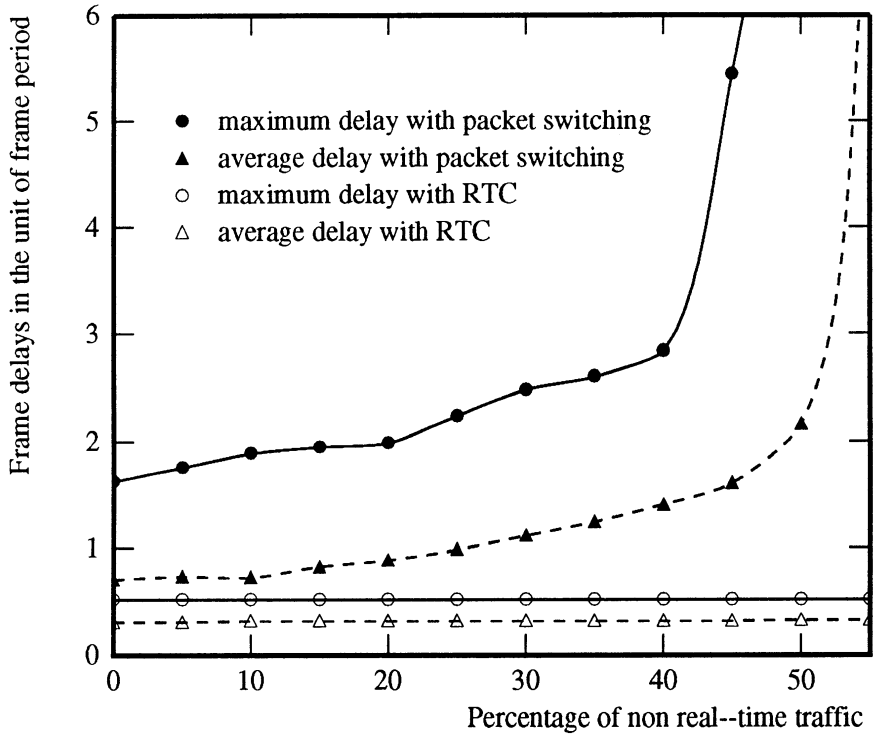

Figure 7.6: Maximum and average video frame delays with Real–time Channels (RTC) and ordinary packet switching.

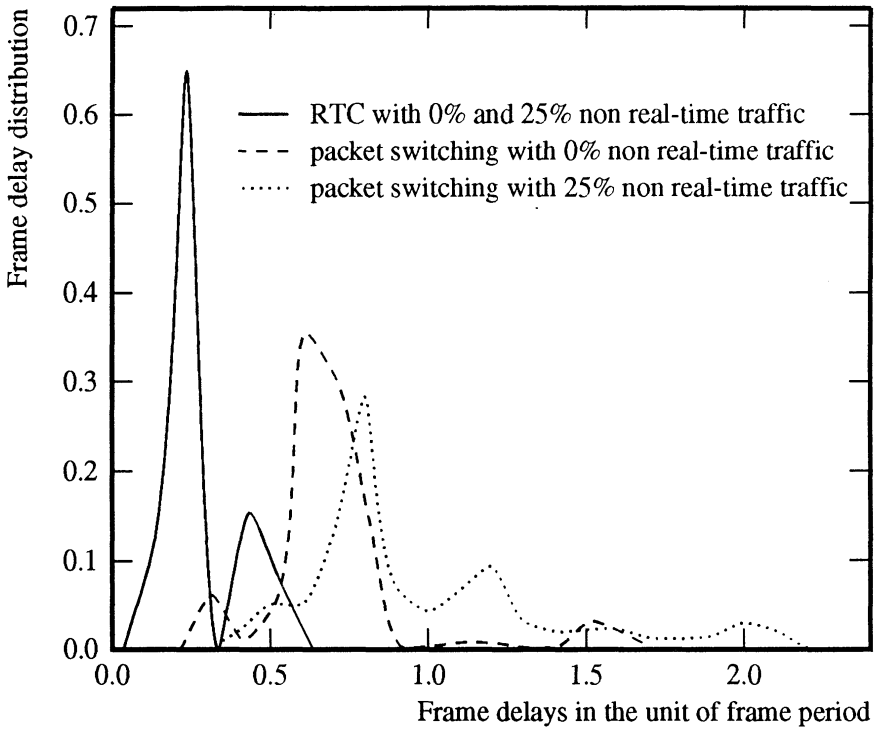

Figure 7.7: Frame delay distributions. The figure shows the number of frames, normalized by the total number of frames transmitted, whose transmission delays lie in each interval equal to a 1/10 of the frame period.

With the ordinary packet switching, on the other hand, frame delays increase as non real–time traffic increases. This makes the transmission of real–time video signals very unreliable. Notice that several large file transmissions could easily get the network congested. During such a transmission period, the frame delays could exceed 10 frame periods, interrupting the receiving side even if a few frame buffers are used.

Adding a priority mechanism to the ordinary packet switching, i.e., giving real–time packets a higher transmission priority than non real–time packets, could alleviate the problem. However, the performance still will not match that of real–time channels for the following reasons.

1. There exist differences in urgency among real–time messages. For example, the messages of channel 0 (which need to travel 19 links) are more urgent than those of other channels (which only travel at most 3 links). Also, some nodes may have larger frame buffers than others, and thus the frames sent to these nodes can tolerate larger transmission delays. A priority mechanism usually does not distinguish these differences. This can be seen in Figs. 7.6 and 7.7. In the absence of non real–time messages, the frame delays of channel 0 with packet switching are still larger than that with real–time channels. The reason for this phenomenon is that when real–time channels are established, Protocol 3.3 gives channel 0 smaller link delays than that of the other channels. Thus, the packets of channel 0 will be assigned tighter deadlines and are thus more likely to be transmitted before the packets of other channels. In conclusion, the real–time channel protocol is more flexible than the priority mechanisms in accommodating heterogeneous real–time traffic (i.e., with different transmission delay and bandwidth requirements).

2. No efficient traffic control schemes are known for (priority) packet switching. Thus it is still possible that real–time traffic could temporarily congest the network. The real–time channel protocol, on the other hand, will avoid congestion by rejecting requests for establishing new channels at the channel establishment stage.

3. Even if a traffic control scheme is employed for (priority) packet switching, the established channels could still affect each other. A video channel, for example, could create more traffic than specified from time to time (say, when the scene moves very fast). This extra traffic could affect the timely transmission of other channels' messages. As discussed in Chapter 3, this will not happen to the real–time channels due to the deadline scheduling of packet transmissions.

7.3 FDDI-M

We showed in the previous section that real–time channels outperform both the circuit–switched and packet–switched transmission. However, implementation of real–time channels requires significant hardware and software changes to the existing computer networks. In local area ring networks like the FDDI, an easy way to improve the networks' multimedia supporting capacities is to use the modified protocol, FDDI-M, proposed in Chapter 6.

We carry out extensive simulations to verify the advantages of the FDDI-M over FDDI and FDDI-II. The network simulated is a medium–size FDDI single ring of 50 nodes and 92-kilometer ring length. Assuming a node latency of 0.6 microsecond and a propagation delay of 5.085 microseconds per kilometer [30], the ring has a latency T_{ring} = 0.5 ms.

According to the FDDI standard, the maximum-packet size is 36 Kbits, and thus, the maximum packet transmission time $C_p = 0.36$ ms.

The synchronous traffic under consideration is the digital motion video signals compressed with the MPEG [44]. Each MPEG video channel has an average signal rate of approximately 1.5 Mbps and the video frames are transmitted at the rate of 30 frames/second. Thereby the frame generation period $T = 33$ ms. Each video frame is required to be transmitted before the generation of the next frame. The MPEG compression algorithm generates a large frame, called a *prime frame*, for every eight frames. The size of a prime frame could be three times as large as the average frame size. In our simulation, the prime frames are generated with their sizes uniformly distributed in the range of 100 Kbits $\sim$ 150 Kbits, and other frames are generated with their sizes uniformly distributed in the range of 25 Kbits $\sim$ 75 Kbits.

Asynchronous messages are generated randomly with their sizes uniformly distributed in the range of 0 $\sim$ 1.5 Mbits by those 10 nodes which are not transmitting video signals. To test a ring's ability of accommodating asynchronous traffic, the asynchronous traffic is generated with a total rate equal to the ring's bandwidth of 100 Mbps. So, the actual throughput represents a ring's ability of accommodating asynchronous traffic. The fairness of a protocol can also be verified by checking each node's asynchronous traffic throughput.

Three protocols, FDDI, FDDI-II, and FDDI-M, are compared for their ability to support video channels and asynchronous traffic. The configurations of these protocols are described below.

FDDI: Two configurations are simulated: (1) $TTRT = T/2 = 16.5$ ms, and (2) $TTRT = T = 33$ ms. The reason for using the second configuration is to check if it is really necessary to set the TTRT to one half of the requested delay bound. For both configurations, the high priority token holding time of a node that generates video signals is set to the time needed to transmit a maximum-size frame h_i = 150 Kbits / 100 Mbps = 1.5 ms. Theoretically, no more than $n = (TTRT - T_{ring} - T_p)/h_i$ video channels should be established to satisfy the protocol constraint $\sum_{i=0}^{n-1} h_i \leq TTRT - T_{ring} - T_p$ (see Chapter 6). This results in $n = 10$ for $TTRT = 16.5$ ms, and $n = 21$ for $TTRT = 33$ ms. In our simulation, however, when more than n video channels are requested, we ignore this constraint to see how many frames will not be transmitted before the generation of the next frame.

FDDI-II: In order to ensure each video frame to be transmitted before the generation of next frame, a transmission bandwidth of 150 Kbits / 33 ms = 4.5 Mbps is needed by each video channel. Thus, 98/4.5 = 21 video channels can be established. When $n > 21$ video channels are requested, we divide the 98 Mbps bandwidth equally among the channels. In this case, some frames may need longer than 33 milliseconds for their transmission.

FDDI-M: The configuration of the FDDI-M is the same as that of the FDDI except that the ring's TTRT is set to $33 - 1.5n$ ms, where n is the number of video channels established. Since the TTRT should not be smaller than $T_{ring} = 0.5$ millisecond, up to $n = (33 - 0.5)/1.5 = 21$ video channels can be established with the FDDI-M. When $n > 21$ video channels are requested, we set $TTRT = 2$ ms to guarantee a certain level of asynchronous throughput.

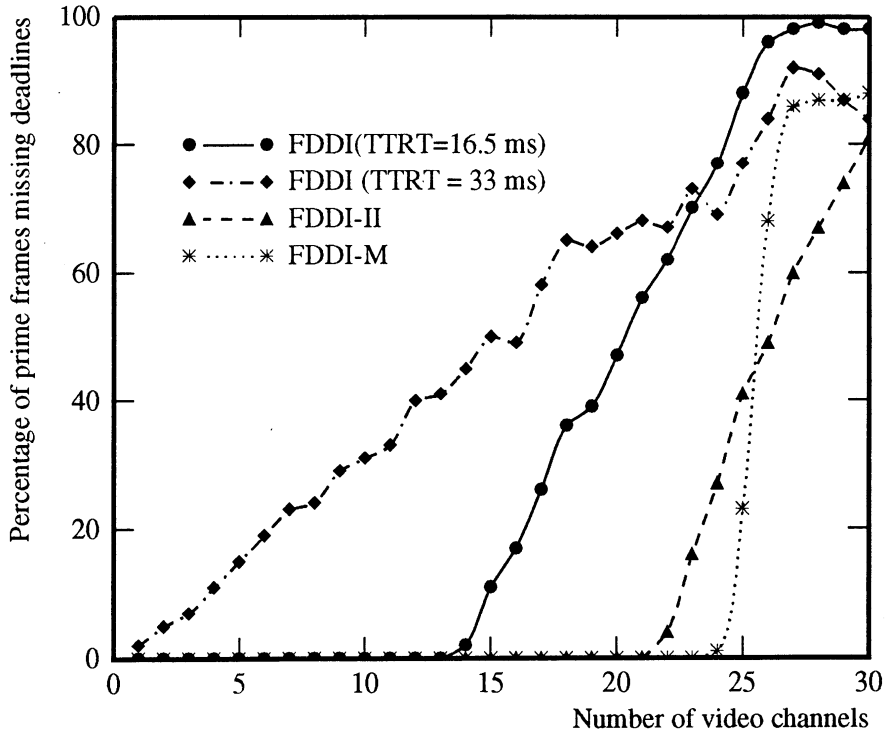

Figure 7.8: Percentages of prime frames missed deadline.

In our simulation, we use nodes $0 \sim (n-1)$ as the source nodes of the n requesting video channels. Asynchronous messages are generated by nodes $35 \sim 44$. For each channel number $n = 0, 1, \cdots, 30$, we used a simulation period of 100 seconds during which each video channel generates 3300 frames. To simulate the worst case, all video channels start transmitting frames at time 0. For each n, the percentages of channel $(n-1)$'s prime frames missing their deadlines under different protocols are plotted in Fig. 7.8. We considered the prime frames only since they are more important than other frames (decompression of other frames needs information from the prime frames). Also, the prime frames are more likely to miss their deadlines because of their large size.

From Fig. 7.8, up to 13 video channels can be established in an FDDI ring with $TTRT = 16.5$ ms without any late frames. Setting up any more channels would cause the loss of prime frames. The reason why a few more than the calculated 10 channels can be established is that the worst case which causes the largest token rotation time did not happen in our simulation. The token will experience the largest rotation time if (1) no packets were transmitted during the previous token's rotation, and (2) the maximum number of synchronous and asynchronous packets are transmitted during the current token's rotation. In our simulation, since nodes $35 \sim 44$ always have many asynchronous packets to transmit, (1) will never happen. Setting $TTRT := 33$ ms for the FDDI does not work at all. The prime frames missed their deadlines even in case of a small number of video channels. This means that the token rotation time could exceed the TTRT even if the synchronous traffic is light. Thus it is necessary to set the TTRT to one half of the requested delay bound in an FDDI network.

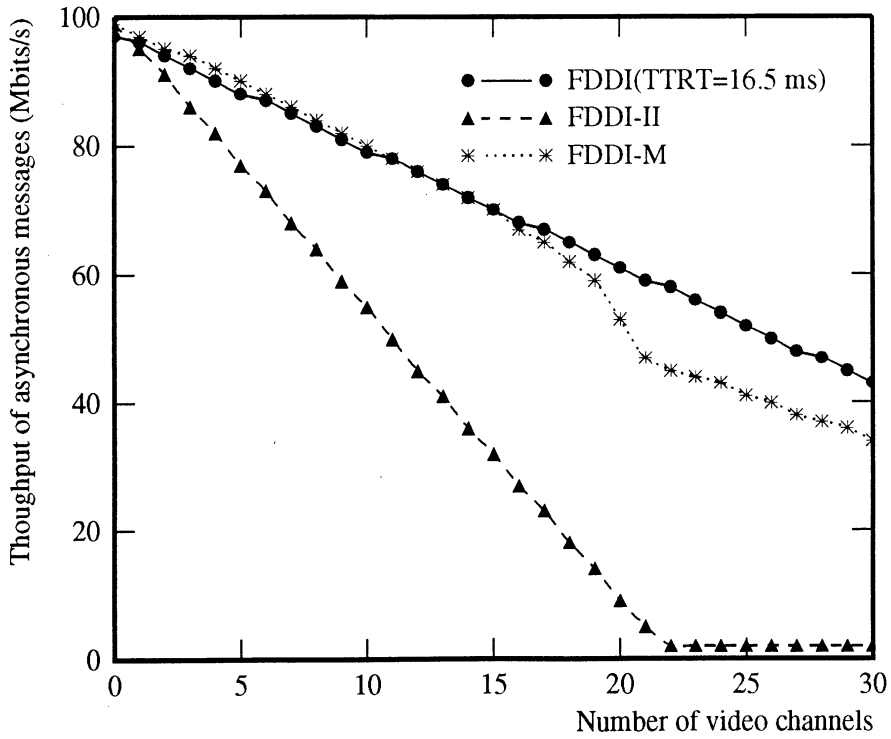

Figure 7.9: Throughput of asynchronous traffic.

As expected, the FDDI-M outperforms the FDDI by far. It can accommodate 24 video channels which is 11 channels more than that of the FDDI ($TTRT$ = 16.5 ms). The FDDI-II behaves exactly as calculated. Up to 21 channels can be established. The reason why the FDDI-M performs a little better than the FDDI-II is that the former shares ring bandwidth among different video channels, while with the latter, one channel cannot use the reserved bandwidth of another, even when that channel is not transmitting anything.

The throughput of asynchronous messages are plotted in Fig. 7.9. The FDDI-M has a higher asynchronous throughput when the number of channels is less than, or equal to, 13. This can be seen more clearly in Fig. 7.10 which shows the average token rotation times. As discussed earlier, the smaller the average token rotation time, the more transmission bandwidth is used for the token passing, thus leaving less bandwidth for asynchronous messages. The difference between asynchronous throughput becomes more pronounced with the increase of ring latency (i.e., increase of ring size).

When $n \geq 14$ video channels are established, the FDDI has a higher asynchronous throughput than the FDDI-M. However, this does *not* mean that the FDDI is superior to the FDDI-M. The FDDI's gain in asynchronous throughput is achieved at the cost of missing synchronous messages' deadlines. It is like setting a very large TTRT to have the maximum bandwidth without considering synchronous messages at all. The FDDI-M reduces the average token rotation time to accommodate more video channels. When the number of video channels reaches 21, the FDDI-M's TTRT is fixed at 2 ms. Thus, the average token rotation time no longer decreases, and synchronous messages start missing their deadlines.

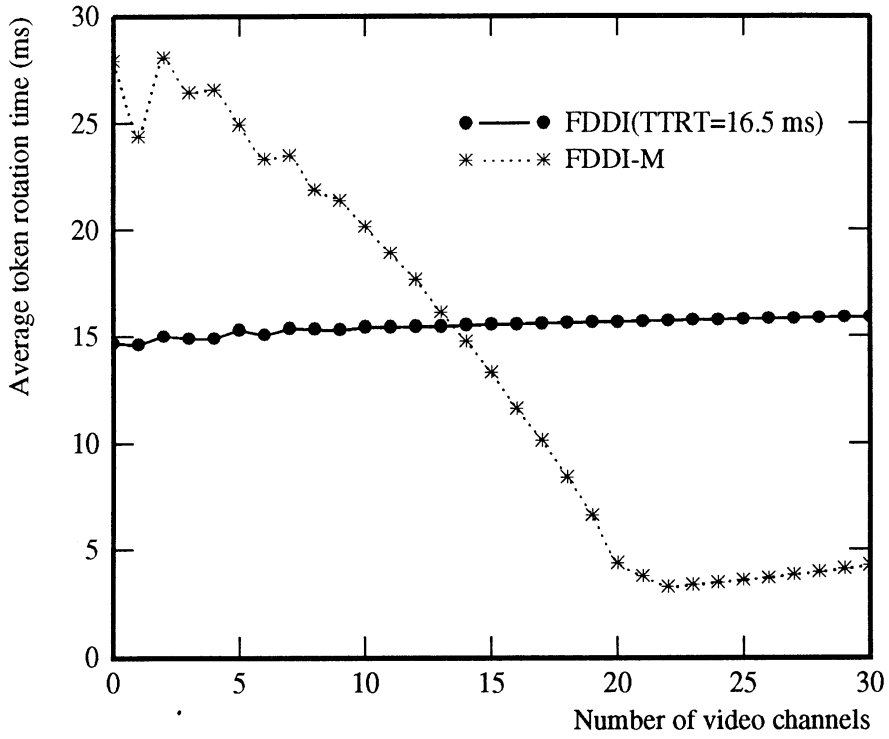

Figure 7.10: Average token rotation time.

The FDDI-II provides the lowest throughput for asynchronous messages. Establishment of each video channel deprives 4.5 Mbps bandwidth from the network. So, after establishing 22 video channels, less than 2 Mbps transmission bandwidth is left for asynchronous traffic. This indicates one of the disadvantages of the FDDI-II when a network is heavily loaded with synchronous traffic.

7.4 Conclusion

With simulations, we show in this chapter the advantages of using real-time channels and the FDDI-M for multimedia applications. Multimedia networking is only one the the many applications which will benefit from real-time communications. We expect a lot more real-time services will be available with the improved quality of service of computer networks.

CHAPTER 8

SUMMARY AND FUTURE WORK

8.1 Summary of the Thesis

This thesis has treated a new, increasingly important subject of computer networking: real–time channels. The main contributions are summarized below.

Deadline scheduling theory: The deadline scheduling theory developed in Chapter 2 includes the sufficient and necessary channel schedulability conditions and an efficient algorithm for the calculation of minimum delay bounds with the deadline scheduling of message transmissions. It paves a solid mathematical basis for the whole real–time channel concept. Compared to the previously obtained results [4, 6], our results allow us to accommodate heterogeneous real–time channels and fully utilize the network's transmission bandwidth.

The deadline scheduling theory thus obtained is also a significant generalization of the classical periodic task scheduling theory developed for a single processor [5]. It removes the restriction that the deadline of each task always equal the generation time of the next task, thereby making the theory applicable to a larger number of areas including the deadline scheduling of message transmissions.

Real–time channel protocol: The real–time channel protocol presented in Chapter 3 is the first complete specification of the real–time channel transfer mode. It is composed of two parts:

1. Message transmission protocol. The message transmission protocol specifies the packetization of real–time messages and calculation of the packet deadlines over each transmission link. The proposed protocol ensures the correctness, solvability, and protection of real–time channels.
2. Channel establishment protocol. This protocol establishes real–time channels using the deadline scheduling theory of Chapter 2. Both the distributed and centralized versions of the protocol are presented.

Deadline scheduler: The need for a fast deadline scheduler for the implementation of real–time channels has been largely overlooked. We show in Chapter 4 the importance of this issue and worked out a design of the scheduler which is the first step of our project on building a prototype system capable of supporting real–time channels.

Fault–tolerance: Simultaneous realization of real–time and fault–tolerant communication is a difficult problem and little has been reported on this issue. With the real–time

channels, we show in Chapter 5 how this problem can be tackled by exploring the spatial redundancy of the network topology.

Local area networks (LANs): The importance of real–time communication over LANs cannot be over emphasized since almost all end–systems are connected to a LAN. We show in Chapter 6 how real–time channels can be established over LANs with a proper MAC protocol like the FDDI. A modification to the FDDI, called the FDDI-M, has also been proposed which can significantly improve an FDDI network's capacity of supporting real–time traffic.

Multimedia applications: To illustrate the power of real–time channels, we investigate in Chapter 8 the application of real–time channels for multimedia networking through simulations, and show how the real–time channels outperform both the circuit–switched and packet–switched transmissions. We have also shown the superiority of the new MAC protocol FDDI-M over the existing FDDI and FDDI-II in supporting video transmission.

8.2 Future Work

The work of the thesis can be followed up in the following three directions.

Prototype implementation: To verify the results of real–time fault–tolerant communication approaches with real–time channels, we need to build and test a prototype communication system. A key problem is an efficient implementation of the deadline scheduling of message transmissions. We have proposed in Chapter 4 a preliminary design of a fast deadline scheduler. The next step is to refine and implement the design.

The HARTS [12] currently being built at the Real-Time Computing Laboratory is an ideal platform to perform experiments on real–time channels. The point–to–point connected hexagonal topology of HARTS will allow us to conduct experiments ranging from the basic real–time channels to the highly reliable IFI real–time channels. After equipping it with fault–tolerant real–time channels, HARTS will become a very promising architecture for critical real–time applications.

Extension to the ATM: This thesis assumes the environment of computer networks for the establishment of real–time channels. With the development of BISDN [1], the current telecommunication network will also carry data traffic. The ATM is expected to become the main transfer mode for the BISDN. Thus, it is important to investigate the suitability of the real–time channel concept for the ATM.

Our preliminary investigation shows that real–time channels and the ATM bear many similarities. Both of them are connection–oriented to provide a proper quality of service and use statistical packet multiplexing to achieve high transmission efficiency. However, the procedures used by real–time channels for channel establishment and message transmission are more sophisticated than those for the ATM in order to achieve *guaranteed* end–to–end delay bounds. By applying the results obtained in this thesis to the ATM, we expect to improve the quality of service provided by the current BISDN networks and expand the application domains of the real–time channels.

Implementation of the FDDI-M: The concept of real–time channels is very attractive, but making the current computer networks capable of accommodating real–time channels needs considerable changes to the network hardware and software. By contrast, upgrading an FDDI network to the FDDI-M requires minimal modifications and can be done transparently to the users. Thus, the FDDI-M is an attractive way to improve local area networks' ability of supporting real–time traffic.

At present, FDDI's capacity of supporting synchronous traffic is capitalized due to the lack of application demands and a proper Station Management (STM) protocol dealing with the synchronous bandwidth allocation and connection establishment. With the rapid advance in multimedia systems, however, we expect computer networks to carry more synchronous traffic. Thus, instead of developing the synchronous MAC circuits and the STM protocol for the standard FDDI, it is better to work directly on the FDDI-M.

BIBLIOGRAPHY

[1] M. Prycker, *Asynchronous transfer mode: solution for broadband ISDN*, Ellis Horwood Limited, Chichester, West Sussex, PO191EB, England, 1991.

[2] D. Ferrari, "Client requirements for real-time communication services," Technical Report TR–90–007, International Computer Science Institute, Berkeley, March 1990.

[3] R. McFarland, "Protocols in a computer internetworking environment," *Proceedings of EASCON 79*, 1979.

[4] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal on Selected Areas in Communications*, vol. SAC-8, no. 3, pp. 368–379, April 1990.

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, January 1973.

[6] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multi-hop networks," in *Proc. 11th Int. Conf. on Distributed Computer Systems*, pp. 300–307. IEEE, May 1991.

[7] Q. Zheng and K. G. Shin, "On the ability of establishing real–time channels in point–to–point packet–switched networks," *IEEE Transactions on Communication* (in press), 1993.

[8] Q. Zheng and K. G. Shin, "Real–time communication in local area ring networks," in *Conference on Local Computer Networks*, pp. 416–425, September 1992.

[9] A. Indiresan and Q. Zheng, "Design and evaluation of a fast deadline scheduling switch for multicomputers," RTCL working document, December 1991.

[10] Q. Zheng and K. G. Shin, "Fault–tolerant real–time communication in distributed computing systems," in *in* Proc. 22nd Annual International Symposium on Fault–tolerant Computing, pp. 86 – 93, 1992.

[11] Q. Zheng and K. G. Shin, "Establishment of isolated failure immune real–time channels in HARTS," submitted for publication, 1992.

[12] K. G. Shin, "HARTS: A distributed real–time architecture," *IEEE Computer*, vol. 24, no. 5, pp. 25–36, May 1991.

[13] K. G. Shin and Q. Zheng, "FDDI-M: a scheme to double FDDI's ability of supporting synchronous traffic," submitted for publication, 1992.

[14] K. G. Shin and Q. Zheng, "Mixed time-constrained and non-time-constrained communications in local area networks," *IEEE Transactions on Communication* (in press), 1993.

[15] Q. Zheng, K. G. Shin, and E. Abram-Profeta, "Transmission of compressed digital motion video over computer networks," to appear in COMPCON Spring'93, 1993.

[16] D. Peng, *Modeling, assignment and scheduling of tasks in distributed real-time systems*, PhD thesis, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI, 1989.

[17] W. J. Dally and C. L. Seitz, "The torus routing chip," *J. Distributed Systems*, vol. 1, no. 3, pp. 187–196, 1986.

[18] D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, M. I. T. Press, Cambridge, Massachusetts, 1987.

[19] J. W. Dolter, P. Ramanathan, and K. G. Shin, "A microprogrammable VLSI routing controller for HARTS," in *International Conference on Computer Design: VLSI in Computers*, pp. 160–163, October 1989.

[20] H. Ahmadi and W. E. Denzel, "A survey of moden high-performance switching techniques," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 7, pp. 1091–1102, September 1989.

[21] F. A. Tobagi, "Fast packet switch architectures for broadband integrated services digital networks," *Proceedings of the IEEE*, vol. 78, no. 1, pp. 133–167, January 1990.

[22] H. Suzuki, H. Nagano, and T. Suzuki, "Output-buffer switch architecture for asynchronous transfer mode," in *Proceedings of International Conference on Communications*, pp. 4.1.1 – 4.1.5, June 1989.

[23] D. D. Kandlur, *Networking in Distributed Real-Time Systems*, PhD thesis, University of Michigan, 1991.

[24] K. G. Shin, "HARTS: A distributed real-time architecture," *IEEE Computer*, vol. 24, no. 5, pp. 25–35, May 1991.

[25] M.-S. Chen, K. G. Shin, and D. D. Kandlur, "Addressing, routing and broadcasting in hexagonal mesh multiprocessors," *IEEE Trans. Computers*, vol. 39, no. 1, pp. 10–18, January 1990.

[26] A. M. Farley, "Networks immune to isolated failures," *Networks*, vol. 11, pp. 255–268, 1981.

[27] A. Tanenbaum, *Computer Networks*, Prentice-Hall, Englewood Cliffs, N.J., 1989.

[28] W. Stallings, *Data and Computer Communications*, Macmillan Publishing Company, New York, second edition, 1988.

[29] F. E. Ross, "An overview of FDDI: The fiber distributed data interface," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 7, pp. 1043 – 1051, September 1989.

[30] A. C. Weaver, "Local area networks and busses – an analysis," Technical report. Flight Data Systems, NASA–Johnson Space Center, 1986.

[31] *FDDI Station Management (SMT) – draft proposed.* American National Standard. ANSI X3T9/92-067, June 25, 1992.

[32] K. C. Sevcik and M. J. Johnson, "Cycle time properties of the FDDI token ring protocol," *Technical Report CSRI-179, Computer Science Research Institute, University of Toronto*, 1986.

[33] G. Agrawal, B. Chen, W. Zhao, and S. Davari, "Guaranteeing synchronous message deadlines with the timed token protocol," in *Proceedings of IEEE International Conference on Distributed Computing Systems.* IEEE, June 1992.

[34] B. Chen, G. Agrawal, and W. Zhao, "Optimal synchronous capacity allocation for hard real–time communications with the timed token protocol," in *Proc. Real-Time Systems Symposium.* IEEE, December 1992.

[35] R. M. Grow, "A timed token protocol for local area networks," in *Electro 82*, May 1982.

[36] M. J. Johnson, "Fairness of channel access for non–time–critical traffic using the FDDI token ring protocol," *Proc. seminar real–time local area networks*, pp. 145–157, 1986.

[37] M. Liebhold and E. M. Hoffert, "Toward an open enviroment for digital video," *Communication of ACM*, vol. 34, no. 4, pp. 104 – 112, April 1991.

[38] E. A. Fox, "Advances in interactive digital multimedia systems," *Computer*, pp. 9 – 21, October 1991.

[39] B. I. Szabo and G. K. Wallace, "Design considerations for JPEG video and synchronized audio in a Unix workstation enviroment," in *proceedings of summer 1991 Usenix Conference*, pp. 353 – 368, 1991.

[40] T. D. C. Little and A. Ghafoor, "Network considerations for distributed multimedia object composition and communication," *IEEE network magazine*, pp. 32 – 49, November 1990.

[41] M.-S. Chen, Z.-Y. Shae, D. D. Kandlur, T. P. Barzrilai, and H. M. Vin, "A multimedia desktop collaboration system," Research Report, IBM Research Division, T. J. Watson Research Center, 1992.

[42] M. Teener and R. Gvozdanovic, "FDDI-II operation and architectures," in *proceedings of the 14th conference on local computer networks*, pp. 49–61, 1989.

[43] G. K. Wallace, "The JPEG still picture compression standard," *Communication of ACM*, vol. 34, no. 4, pp. 30 – 43, April 1991.

[44] D. Gall, "MPEG: a video compression standard for multimedia applications," *Communications of the ACM*, pp. 46–58, 1991.