Order Number 9500904

# Real-time communication with statistical performance guarantees

Chou, Chih-Che, Ph.D.

The University of Michigan, 1994



U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

# REAL-TIME COMMUNICATION WITH STATISTICAL PERFORMANCE GUARANTEES

by

Chih-Che Chou

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1994

Doctoral Committee:
Professor Kang G. Shin, Chair
Professor James C. Bean
Associate Research Scientist Chinya V. Ravishankar
Assistant Professor Stuart Sechrest
Professor Toby J. Teorey

To My Parents, and Fan-Lu

# ACKNOWLEDGEMENTS

I am most grateful to my advisor, Professor Kang G. Shin, for his guidance and constant encouragement throughout my graduate school years, which made my Ph.D. process so much less overwhelming and will continue to lead me in the future. He introduced the field of real-time communication systems to me four years ago, and has always been eager to discuss our topics with invaluable technical and professional advice. Many thanks also go to my committee members for their important comments and suggestions.

This dissertation work has been supported in part by National Science Foundation under Grant No. MIP–9203895 and the Office of Naval Research under Contract No. N00014–92–J–1080. I would like to thank Gary Workman and Ben Gross of General Motors for their technical and financial support for the work described in Chapter 6.

I appreciate all fellow students of the Real-Time Computing Laboratory, especially Hagbae Kim, Ashish Mehra, Lup Houh Ng, Atri Indiresan, Jennifer Rexford and Seok-Kyu Kweon, for their friendship and insightful discussions. I also want to thank many friends on the campus by whom my life in Ann Arbor could be decorated with many unforgettable memories.

Although they are in the distance across the Pacific Ocean, my parents and sisters have bestowed a constant love and encouragement on me. My study could not be possible without their support. Finally, I would like to express my gratitude and love to my wife, Fan-Lu, without whose support and patience this dissertation would have never been completed.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF APPENDICES

# CHAPTER 1

# INTRODUCTION

## 1.1 Motivation

There has been an increasing need for timely, dependable communication services for such embedded real-time applications as aircraft, intelligent vehicles, automated factories, industrial process controls and many multimedia applications. Such applications are usually realized by executing a number of cooperating/communicating computational tasks on multiple processors before their deadlines imposed by the corresponding mission/function. To support the timely execution of these tasks, the associated communication subsystem must guarantee the delivery of time-critical messages before their deadlines (defined by the deadlines of communicating tasks). Unfortunately, the OSI standards in their present form cannot adequately support time-critical applications, although some of them provide a limited form of real-time communication services. Since the network architectures that have been standardized so far are designed primarily for general-purpose, non-real-time traffic, they cannot always provide adequate resilience and performance for time-critical applications, especially when real-time and non-real-time traffic coexist.

The term "time-critical" or "real-time" is used to represent the presence of a time window, within which one or more specific actions are required to be completed with some pre-defined level of certainty and some pre-defined level of quality. Failure to complete specified operations within the time window risks failure of the application that needs the correct and timely execution of these operations, or even worse, loss of plants, vehicles, and possibly human lives. While the agreement about the presence of time window generally exists, the duration of the window, and the required level of certainty and quality remain an application-dependent issue. For this reason, it is not feasible to define or implement a universally optimal network protocol for all applications. However, in addition to highly

reliable media and signaling methods, i.e., achieving a very low bit-error rate and a minimum number of retransmissions, any network supporting real-time communication must satisfy the following requirements [4].

R1. Both time-critical and non-time-critical services should be provided.

R2. Network performance changes resulting from such dynamic factors as load changes and unrelated component failures should not compromise the guaranteed quality of service.

R3. A network should be able to support efficient cyclic or repetitive polling or data transmission. This requirement is necessary in virtually all real-time applications, e.g., audio and video messages, sensor sampling, and actuator signals.

R4. Dynamic adaptation to the change of traffic load and pattern is required. In other words, the ability of independently adding and deleting real-time connections is necessary to make efficient use of a network. Such addition and deletion must not compromise the quality of existing real-time services.

R5. A network should be able to provide a certain form of fault-tolerance according to user/application requirements.

R6. A protocol should allow network designers to fine-tune the network parameters for their specific applications.

R7. Whenever possible, a network must be compatible with the existing protocols and networks (for economic reasons).

Obviously, any network with the above features will provide a very general form of real-time communication with a guaranteed quality of service for various applications and will also provide the flexibility to adapt itself to different applications. This dissertation develops and evaluates a communication subsystem that satisfies all of the above requirements.

Although real-time communication with a guaranteed quality of service is important and necessary for many embedded real-time systems, it has not yet been addressed thoroughly. Real-time communication between control and sensing devices in an automated factory is a typical example of such applications. In an automated factory, various devices such as robots, sensors, transport mechanisms and control computers are connected by a network,

and the ability to communicate with each other in a predictable manner is of great importance. The Manufacturing Automation Protocol (MAP), which was proposed by GM and other companies for automated factory networks, is based on the OSI seven-layer model and the token bus protocol, IEEE 802.4. Although the MAP can provide some limited form of real-time communication, it cannot provide any guarantee of delivering messages before their deadlines. In fact, the MAP only provides some temporal ordering between devices based on their priorities. The seven-layer MAP is usually too slow to be used for real-time communication, since there are at least 14 layers' delay in a single one-way message transmission. Another protocol called MINIMAP employs only the first two layers of MAP and combines the remaining five layers into a single layer. Although the communication delay is expected to decrease with MINIMAP, the real-time issues remain unanswered. In fact, other OSI standards are not adequate for such automated factories either. Token ring type protocols cannot be used for the same reason as the token bus. CSMA/CD type protocols are not applicable to real-time communication because of their unbounded communication delays.

Multimedia applications are another important category of applications which need predictable real-time communication services. Although most of them require only a "statistical" type of performance guarantees, i.e., certain percentage of frame loss can be tolerated, they do need predictable real-time communication to guarantee the application's quality. When dealing with these applications, most researchers focus on compression algorithms and run-time scheduling or admission control issues for the currently available protocols and technologies. However, since these protocols were primarily designed for non-real-time traffic, they cannot provide any predictable guarantees. From these two examples, we can see the need for a protocol which can provide predictable real-time communication services.

## 1.2 Literature Survey and Research Objectives

The problem of supporting time-constrained communication has been studied by several researchers, since it plays a key role in many real-time applications. Most of these efforts can be divided into two categories. The first category is mainly concerned with designing medium access protocols for multiaccess networks while considering time constraints in delivering messages. In this context, most of the proposed schemes can be classified as *best-effort* schemes, where the system tries to ensure that most messages can meet their

deadlines, but it cannot give any guarantees of delivery delays [8,9,26,36,41]. However, based on the given information about the message arrival/generation pattern, some best-effort schemes can make guarantees about their delivery time [40].

The other category deals with establishing real-time point-to-point channels and giving absolute guarantees of maximum delivery delays [16,18,22–24,31,38,44]. Among them, the concept of "real-time channel" proposed by Ferrari and Verma [18] can be used to provide real-time communication services with performance guarantees in wide-area point-to-point networks. A real-time channel is a unidirectional virtual circuit which can provide real-time communication with performance guarantees for given user requirements and traffic specifications [18]. The main issues addressed in these schemes are message scheduling, buffer management, and flow control in the network nodes. In addition to the concept of real-time channels, most of these schemes assume the traffic generated by a real-time channel can be described by a modified version of "linear bounded model," originally proposed by Cruz [15]. In this thesis research, we also use the concept of real-time channels and adopt a modified (more generalized) version of the linear bounded model.

Although the schemes in this category can provide real-time communication with performance guarantees under certain traffic assumption, there are two serious issues which remain unanswered. The first issue is the level of performance guarantees. As we discussed in Section 1.1, the required level of quality and certainty of performance is an application dependent issue. Since different applications may need different levels of service quality, and many applications may tolerate a certain level of data loss, absolute performance guarantees are *not* always necessary. Therefore, the network service provider needs to provide not only the hard real-time communication services but also real-time communication with *statistical* performance guarantees which can be specified by users.

The second issue is the inadequacy of point-to-point network environment. Although a point-to-point network with an arbitrary topology is a very general form of interconnecting hardware structure, it is not suitable for many applications, especially those applications requiring a large number of nodes in a small area, e.g., a workcell in an automated factory, a vehicle and a campus/enterprise network. A point-to-point network is not adequate or cost-effective for such applications due to the complexity of connecting hardware and the potentially long delivery delay. Local area network (LANs), such as multiaccess buses or token rings, are suitable candidates for such applications. They are simple, economical, and have propagation/delivery delays. These advantages make multiaccess networks a very

good candidate as the underlying architecture for small-area real-time applications. However, due to the difference of hardware architecture, schemes which are suitable for wide-area point-to-point networks may not be appropriate for multiaccess LANs. In a point-to-point network, the node at the end of a link has complete control of the link at run-time, and thus, can adopt any scheduling algorithm and utilize the entire bandwidth of the link, if needed. By contrast, nodes which are attached to a LAN need to cooperate with each other in order to transmit data/control packets, since only one node at a time can transmit packets on the shared medium. Therefore, we must develop a scheme which can provide real-time communication with performance guarantees in a LAN environment for such applications. We will propose a scheme which can provide both hard and statistical real-time communication for a multiaccess network.

On the other hand, although LANs can be used in many applications and many end systems are connected to LANs, they are limited to a relatively small area. In order to support real-time communication between nodes which are not connected directly by a multiaccess network, we need schemes which can connect LANs with each other or connect multiple LANs with a point-to-point network via bridges or routers.

Generally, two distinct phases are required to realize the concept of multi-hop real-time channel: off-line channel establishment and run-time message scheduling. The channel establishment phase is of prime importance to the realization of a real-time channel, and during this phase, the system has to select a route between the source and the destination of the channel along which sufficient resources must be reserved to meet the user-specified delay and buffer requirements. Although most of the schemes for point-to-point networks mentioned above can be used in this case, the route-selection problem has not yet been treated in depth.

Since the number of possible routes between two communicating peers could be large, selecting a route for each real-time channel is potentially a time-consuming task. It is therefore very important to develop an efficient scheme that is guaranteed to select a "qualified" route, if any, for each real-time channel request. If the worst-case anticipated traffic over a real-time channel is given, a "qualified" route for this channel is defined to be the one that can meet the user-specified end-to-end delay requirement without compromising any of existing guarantees.

Many schemes assume a global network manager to have knowledge of the resource distribution/utilization and all existing real-time channels in the network [22–24,38]; so

the network manager can solve the route-selection problem for all other nodes which need real-time channels. However, there are many disadvantages and problems of using a global network manager to handle the route-selection problem. The most significant one is that the network manager becomes a performance and reliability bottleneck of the system. The authority of the global network manager is also an apparent problem, because there may be many different organizations which are all connected to the network. Therefore, in spite of the advantages of a centralized solution, a distributed route-selection algorithm is desirable and necessary for wide-area networks.

Although the route-selection problem for real-time channels is different from the traditional message routing problem for point-to-point networks, we do find some similarities between these two problems. Specifically, we will propose two distributed route-selection schemes which use the concept of the Bellman-Ford shortest path algorithm [9,41,46] and the traditional routing tables in the previous ARPANET routing strategy [9,25,34,35,39, 41]. These two route-selection algorithms can be used with the schemes proposed to support real-time communication on multiaccess networks, and hence, can expand our LAN solution to wide-area networks.

## 1.3 Outline of the Dissertation

The goal of this thesis is to develop a scheme which can provide real-time communication with statistical (including absolute) performance guarantees for both multiaccess networks and point-to-point networks.

In Chapter 2, we propose the concept of statistical real-time channels on a multiaccess network. Based on the user's traffic specification and performance requirements, we develop a *channel-based* scheme which can provide real-time communication services with statistical performance (including absolute) guarantees by reserving and allocating communication resources to each channel independently.

In order to let the system add/delete real-time channels independently, we used a channel-based design in Chapter 2 (i.e., each channel is treated independently) without considering the problem of *multiplexing* real-time channels. As a result, the scheme proposed in Chapter 2 still under-utilizes the network, although its capability of supporting real-time communication is much better than conventional LANs, such as FDDI and token bus. Therefore, based on the channel-based scheme, in Chapter 3 we propose a scheme to

multiplex real-time channels which originate from a same node, and at the same time, preserve the ability of independent addition and deletion of real-time channels. This scheme, called the "*node-based scheme*" as opposed to the channel-based scheme in Chapter 2, can significantly improve the network utilization by reducing the bandwidth reservation to the average level from the worst case.

In order for the channel-multiplexing strategy to work correctly, we need to consider the frame inter-dependency problem which is common for many real-time applications, especially multimedia applications. In Chapter 3, we also propose the *multiple-due-date* scheduling algorithm to solve the frame inter-dependency problem in the run-time scheduling phase. Simulation (using MPEG-coded frames from the movie Star Wars) results are presented and the integrated scheme is shown to be able to effectively improve network utilization and provide a guaranteed frame reconstruction rate.

Chapter 4 and Chapter 5 deal with the problem of route-selection for real-time channels which is the key to extend the proposed LAN solution to the environment which contains multiple interconnected LANs and/or point-to-point networks.

In Chapter 4, we propose a generic distributed route-selection strategy which is designed for general real-time channels and is guaranteed to find a qualified route, if any, for each real-time channel-establishment request. Although this scheme can provide a complete and general solution to the route-selection problem, it often over-estimates link delay when there are simultaneous multiple channel establishment requests and its operational overhead is linearly proportional to the number of links in the network. Hence, in Chapter 5 we propose a table-driven route-selection strategy which is designed to support certain types of real-time channels and can solve the route-selection problem by a simple table look-up. As can be seen in Chapter 5, if we only have to support real-time channels with limited, yet important, types – like interactive video – of traffic-generation behaviors, we can improve the efficiency and performance of the route-selection scheme significantly.

Chapter 6 describes in detail how the proposed scheme can be applied to the new industrial standard ISA/IEC SP-50 FieldBus protocol. We also simulate the proposed scheme with FieldBus under an automated factory environment.

The thesis concludes with Chapter 7 which summarizes our contributions and suggests future research directions.

# CHAPTER 2

# STATISTICAL REAL-TIME CHANNEL ON MULTIACCESS NETWORKS

## 2.1 Introduction

The problem addressed in this chapter is the development of a scheme which can provide statistical performance guarantees for real-time channels in a multiaccess network environment. Performance/deadline guarantees are, by definition, certain grades of service which are promised by the communication system. Such guarantees may be defined by user-specified parameters which are given at the time of setting up real-time communication services. The maximum message delivery delay and the maximum (acceptable) message loss rate are two typical performance parameters. In order to provide performance guarantees, the underlying communication system has to reserve *a priori* certain resources for "anticipated" real-time traffic in order to meet performance requirements.

As discussed in Chapter 1, although using real-time channels to provide real-time communication with absolute performance guarantees for point-to-point networks has been studied by several researchers [18,22–24,38,44], there are many applications which prefer different hardware architectures (LANs) and require only "statistical" performance requirements. In this chapter, we focus on the issue of providing real-time communication services with statistical (including hard) performance guarantees on *multiaccess* local-area networks. Although real-time channels were originally designed for point-to-point networks, we adopt the same terminology for the multiaccess network environment throughout this thesis.

Several schemes for real-time communication on multiaccess networks [26,32,41] have been proposed, or implemented on the token ring and the token bus, but generally belong to the "best-effort" category. They do not reserve resources according to the user-specified traffic characteristics and performance requirements, and have no explicit admission control.

Thus, even if each node is guaranteed to have access to the network within some upper bound of delay, the network still cannot guarantee to deliver all real-time messages in time. In this chapter, we propose a scheme which can provide performance guarantees for real-time traffic on multiaccess networks and, at the same time, improve network utilization.

The chapter is organized as follows. In Section 2.2, we state the problem of providing real-time communication services on multiaccess networks. The proposed solution with absolute performance guarantees is presented in Section 2.3. Section 2.4 deals with the real-time communication with statistical performance guarantees. Run-time scheduling issues are discussed in Section 2.5. We present a simulation for comparison and verification in Section 2.6 and the chapter concludes with Section 2.7.

## 2.2 Problem Statement

There are two main approaches to supporting real-time communication. One is the best-effort approach which does not provide any performance guarantees. The other is the "hard" real-time approach which provides "absolute" performance guarantees, but requires *a priori* reservation of all necessary resources based on the worst-case traffic-generation behavior. Both of them have drawbacks of their own. The former does not provide any performance guarantees at all, and the latter often under-utilizes the reserved resources.

Different applications come with different performance requirements and traffic-generation characteristics. Some applications, such as the conversation between two cooperating robots, may need hard real-time performance guarantees, while many other applications may require less stringent performance guarantees. For example, reading various sensors is a typical task in automated manufacturing systems, and missing some of these readings is tolerable as long as the missing frequency is lower than a pre-specified value. For applications of this type, the best-effort approach is not suitable because it offers no performance guarantees at all, and the hard real-time approach is not suitable either, as it requires more resources to be reserved than actually needed. We propose the concept of *statistical real-time channel* [13, 18] to solve the problem of providing real-time communication with "statistical" performance guarantees for these applications.

A statistical real-time channel is defined as a unidirectional virtual circuit which can provide real-time communication with performance guarantees in statistical terms, e.g., the probability of a packet's delivery before its deadline $D$ is greater than a given number

$Z$. Because of its less strict performance requirements, a statistical channel is likely to reserve *less* resources than its hard real-time counterpart, yet provides acceptable real-time performance. Although this concept was proposed in [18], how to realize it has not yet been studied in depth.

Both how user's traffic characteristics and performance requirements are specified and who (which layer) is responsible for this specification have an important bearing on the statistical channels. The current OSI 7-layer model can at best treat real-time communication as best-effort services, since there is no notion of performance guarantees or traffic specification. Moreover, since there are too many layer-to-layer data conversions, the 7-layer model is not adequate for real-time communication. The MINIMAP [21,43] and the FieldBus [3,38] protocols have only three layers in order to reduce the time required for layer–to–layer data conversions. Under these protocols, a real-time communication system consists of only three layers: the physical layer, the data link/network layer which is responsible for providing performance guarantees, and the application/user layer which deals with all user interfaces as well as user-specified traffic characteristics and performance requirements. Obviously, it is the application/user layer's responsibility to derive various user performance requirements or traffic specifications, since only this layer is aware of user's performance needs and traffic characteristics. The derivation should be independent of the network service provider, i.e., the data link/network layer which is responsible for implementing network communication with performance guarantees. So, the application/user layer is responsible for deriving traffic characteristics and performance requirements, and may choose to regulate user's input traffic [15,24,31,44] in order to honor existing performance guarantees. On the other hand, the data link/network layer is responsible for testing, checking, accepting/rejecting requests for establishing real-time channels and provides performance guarantees by using both the reservation and scheduling schemes which the application/user layer is not usually aware of. Since the derivation of performance requirements depends greatly on the application under consideration, we will not discuss it any further.

We will instead consider the problem of providing real-time performance guarantees at the data link/network layer in multiaccess networks. Generally, there are three types of medium access protocols. The first type is CSMA/CD, such as the Ethernet, which is completely random and cannot provide any guarantee on the maximum access delay and therefore, is not suitable for real-time applications. The second type is the distributed timed-token protocol, such as the token ring and token bus. In this type of protocol, a

token is rotated among all nodes on the network, and only the node possessing the token is allowed to transmit data/control packets. The token rotation is governed by some rules which guarantee each node to have medium access at least once within some specified period. The third type of protocols uses a control unit for medium access control on each multiaccess link/bus. This control unit follows some rules to ensure that user-specified performance requirements will be met. In the next section, we will compare the differences between type 2 and type 3 protocols, and identify the advantages of using type 3 protocols and the disadvantages of using type 2 protocols. We will introduce a link control unit for each multiaccess link which is responsible for controlling and coordinating the link access for all nodes on that link. The link control unit is also responsible for allocating tokens and testing whether or not to admit hard/statistical real-time channels. Each real-time channel is required to reserve a portion of link bandwidth by sending a channel (connection) establishment request to the link control unit before sending its first packet over the link.

## 2.3 Hard Real-time Channels on Multiaccess LANs

In order to provide "statistical" performance guarantees, like

$$P(\textit{delay of a packet} \leq D) \geq \textit{a given } Z \text{ or,} \tag{2.1}$$

$$P(\textit{no packet loss in any time interval of certain length}) \geq Z, \tag{2.2}$$

the communication system needs to know the distribution of packet arrivals in each channel. Those packets missing deadlines are considered "lost." Using the user-specified delay bound $D$, the maximum packet size $S_{max}$, the maximum burst size $B_{max}$, and the maximum packet-arrival rate $G_{max}$, one can establish a hard real-time channel in a point-to-point network using one of the schemes in [18,22,23,47]. Considering the differences between multiaccess local-area networks (LANs) and point-to-point networks, we will first develop a new scheme for establishing hard real-time channels on multiaccess LANs with a more general traffic specification than those used for point-to-point networks. We will then identify the additional information needed to establish statistical real-time channels on a multiaccess LAN.

The primary difference of a channel (connection) in a multiaccess LAN from that in a point-to-point network is the relationship between nodes and links, and the LAN's shorter packet-delivery latency. In a point-to-point network, a node has complete control of its transmission links and has complete knowledge on whether or not a channel running through

the node needs to be scheduled for transmission by simply examining packets in the queues, one for each channel; that is, it is easy to multiplex several real-time channels. However, in a multiaccess LAN, it is difficult to determine which node has the right to transmit at a particular instant, especially in the presence of both real-time and non-real-time packets. There exist several medium access control protocols which can achieve fair medium access among all nodes, and some of them can support a limited form of real-time communication. For example, FDDI uses a target token rotation time (TTRT), a high-priority token holding time (for synchronous packets), and a token holding time (THT) to ensure that the maximum time for synchronous packets to wait for medium access will not be greater than 2×TTRT. Although we can adjust the high-priority token holding time and the algorithm of updating THT for each node using the anticipated load of real-time traffic so as to make a good distribution of transmission capacity, FDDI still has many inherent disadvantages. Two of these disadvantages are most significant for real-time traffic. First, TTRTs must be identical for all nodes on the LAN. This restriction may seriously limit the use of FDDI when heterogeneous real-time traffic is to be handled. If some node requires a very short TTRT, the network utilization may be significantly reduced, as the token has to be rotated around the ring very fast[1] (thus wasting the bandwidth used for token passing). Second, since FDDI has no efficient way to dynamically change TTRT and the high-priority token holding time for each node, it cannot be used in a network where the traffic load of each node changes due to the establishment and/or removal of real-time channels. In fact, most timed-token protocols (FDDI is one of them) suffer these two problems, and they are not adequate for supporting real-time communication which requires dynamic and independent addition/deletion of real-time channels.

Another typical example of LAN which supports real-time traffic is FDDI II. In addition to being a timed-token protocol, FDDI II adds the circuit switching feature to facilitate the transmission of real-time traffic. Although it provides some degree of dynamic allocation of link bandwidth based on 16 wide band channels (WBC) [1,2,33,41,42], FDDI II still suffers the aforementioned two problems. Moreover, FDDI II under-utilizes the network resources because of the use of WBCs which cannot be shared by other traffic even when they are idle.

Before proceeding to describe our scheme, we formally define the maximum token return time (MTRT) and real-time token holding time (RTHT) used in our scheme. The MTRT for

[1] according to the smallest TTRT

a real-time channel is the maximum time interval between two consecutive token allocations to the channel. For example, the MTRT for FDDI is equal to $2 \times TTRT$. During each of its token possessions, the RTHT of a real-time channel is the maximum time the channel is allowed to use for the transmission of its real-time packets.

In order to solve the above two problems, we need an adaptive scheme which allows different real-time channels to have different MTRTs. The ring is not a good topology if we want to allow different MTRTs for different channels or nodes; a **multiaccess link/bus** is a natural candidate in this case. Note that ring structure can be used if we only manipulate RTHT. On the other hand, the need of dynamically changing the MTRT and RTHT for each node suggests use of a centralized control unit on **each** multiaccess link which is responsible for token allocation and admission test upon receiving a request for establishing a real-time channel. The centralized solution is much more efficient and cost-effective than its distributed counterpart for dynamic and independent addition/deletion of real-time channels, because a multiaccess link poses no communication bottleneck, no resource deadlock, no routing problem, and low communication overhead in using a centralized control unit. Besides, a distributed counterpart causes potential incoherence and inefficiency because all changes of MTRTs and RTHTs have to be negotiated/accepted by all nodes. Thus, we will use a centralized control unit, called the *link control unit* (LCU), on a multiaccess link. The LCU is responsible for token allocation and resource reservation for real-time channels running through the link. The LCU will send high-priority (i.e., real-time) tokens and normal (i.e., non-real-time) tokens to all nodes on the link according to their needs and fairness. Only the node which currently possesses a real-time (normal) token is eligible to transmit real-time (normal) packets. There is an expiration-time parameter associated with each token. The node must return the token to the LCU before or at the time the token expires.

Fault-tolerance is an important issue for a centralized approach. For a multiaccess link, fault-tolerance issues may be solved by duplication of the LCU. As all activities on a multiaccess network can be seen by all nodes on the network, we can use a passive LCU to maintain the network status just in case the primary LCU fails. Because of the multiaccess property of the network, this approach induces little additional communication cost in dealing with the problem of LCU failure. Another general problem for a centralized solution is high communication overhead, since all control messages have to be routed to the centralized control unit. In point-to-point networks, this may be a serious problem, because control messages may have to travel several hops to reach the centralized control

unit, thus incurring high communication overhead for the transmission of control messages. However, in a multiaccess network this communication overhead is small, because messages can be received by all nodes on the multiaccess network, including the LCU, in one hop. Therefore, the overhead of sending control messages (including tokens) is approximately proportional to the ratio of token size vs. the lifetime of a token (RTHT). Although this ratio depends significantly on the applications, it is generally small for typical applications, e.g., voice/video communication, since the traffic volume (per token allocation) of these applications is usually much larger than the size of a token, and the life of each of these applications is usually much longer than the time needed to set up a channel (connection).

To provide real-time performance guarantees, the LCU reserves link capacity and allocates a real-time token to each node on a *per-channel basis*. However, in Chapter 3, we will also discuss an alternative in which the token is allocated on a *per-node basis*. The basic idea is to let each real-time channel have its own MTRT and RTHT based on its anticipated real-time traffic load. In our scheme, although the MTRT is defined to be the maximum time interval between two consecutive token allocations, it is approximately the same as the corresponding expected time, as the variance of actual token return time is negligibly small if $MTRT >> RTHT$ (true in most cases, if the network is expected to support many real-time channels at the same time).

Among the several models proposed to describe the traffic generated by a real-time channel, the *linear bounded model* — which was originally proposed by Cruz [15] and also adopted by other researchers [23,24,44] — is one of the most general and practical models. The traffic generated by a real-time channel is said to follow the linear bounded model if the number of packets generated in any interval $T$ is bounded by a linear function of the length of the interval $T$, i.e., $(G_{max}T + B_{max})$, where $G_{max}$ is the maximum packet-generation rate of this channel and $B_{max}$ is the maximum burst size. Using the user-specified delivery-delay bound $D$, one can establish a hard real-time channel in a point-to-point network [23,24,44]. In this chapter, we adopt an even simpler model which requires only two parameters for establishing hard real-time channels:

- $D$ (seconds): the user-specified delivery-delay bound for a message,
- $M$ (packets): the maximum number of packets that can be generated in an interval of length $D$.

This model is more general than the linear bounded model, because it can be applied to

more cases, and the linear bounded model is only a special case of this model (by letting $M = G_{max}D + B_{max}$). As can be seen later, we need additional information for handling statistical real-time channels.

We can derive MTRT and RTHT for each real-time channel by using these two parameters which are provided by the application layer when a request for establishing a real-time channel arrives. Since there is generally an upper bound for the size of a packet in a given LAN, we also assume the transmission time for a maximum-size packet, $P_{max}$, is available. Since the size of a message may be larger than $P_{max}$, we need to break this message into several packets for transmission over the network. From these parameters, MTRT and RTHT can be derived by:

$$MTRT := D \tag{2.3}$$

$$RTHT := M \times P_{max}. \tag{2.4}$$

Although there are many other possible solutions which have smaller MTRTs, we do not choose a smaller MTRT here, because the smaller the MTRT, the more frequently the system has to assign a token to this real-time channel and therefore, the less efficient.

When a node attempts to establish a real-time channel, it has to provide the LCU the requested MTRT and RTHT for this channel as arguments of its real-time channel establishment request. The LCU will then try to reserve the link capacity for this channel by performing the following admission test:

$$\sum_i \left( \frac{RTHT_i + overhead_i}{MTRT_i} \right) \leq 1 \tag{2.5}$$

where the index $i$ runs over all existing real-time channels and the current request. The main part of overhead is determined by the token passing time. If the admission test can be satisfied after adding this new channel, the LCU will reserve the required link capacity, update the information about the existing real-time channels, and send a confirmation message to the requesting node. Otherwise, the LCU will send a rejection message to the requesting node. Using these parameters, the LCU can use the deadline-driven scheduling algorithm in [29] to allocate tokens to each real-time channel with the corresponding guaranteed MTRT and RTHT. Basically, the LCU will issue the requesting node a token approximately once every MTRT, thus allowing the node to transmit packets of this real-time channel for a period up to RTHT. If the transmission completes before the RTHT expires, this node has to return the token to the LCU. The LCU will then issue the next token to this node (for

this corresponding channel) no later than ($MTRT - RTHT$) time units after the current token is returned by this node. Thus, if all channels use up all their reserved time, tokens will be issued to all nodes exactly once every (their corresponding) MTRT. The run-time scheduling strategy will be discussed later.

## 2.4 Statistical Real-time Channels on Multiaccess LANs

If the application can tolerate the loss of a certain percentage of real-time packets, using a hard real-time channel which reserves resources for the worst case will severely under-utilize the network. Therefore, a real-time channel which reserves less resources and meets looser requirements is more suitable for such applications. In order to achieve this, we need additional information on packet generation, i.e., the distribution of packet inter-arrival times. Basically, we assume the packet-arrival distribution for a real-time channel (characterized over an interval of length $D$ or MTRT) is given as in Fig. 2.1, and the arrivals in the interval $D$ are independent and identically distributed (*iid*). Then we try to reduce the bandwidth that needs to be reserved for each real-time channel using the packet-arrival distribution and the *iid* assumption. Although this *iid* assumption may seem unreasonable in view of the highly-correlated nature of such real-time traffic as voice and video packets, our simulation results in Section 2.6 show that the *iid* assumption works well for the transmission of compressed (e.g., JPEG [45,48]) digital motion-video frames.

One can lower the bandwidth needed for a real-time channel by increasing MTRT or decreasing RTHT. By increasing MTRT or choosing some MTRT $> D$, we may not be able to satisfy Eq. (2.1) or Eq. (2.2) without making more assumptions, since a packet could still be lost with this larger MTRT even when the packet-arrival rate for this channel is far below average. For example, even though the packet-arrival rate is low, the token may not always arrive in time since MTRT $> D$. Another practical advantage of choosing $MTRT = D$ is the packet-arrival distribution of a real-time channel can be computed off-line, and only one distribution is needed for each application even if it has several different performance requirements, e.g., different packet-loss rates. Therefore, once $D$ is determined, the distribution of packet arrivals from a source can be characterized and derived from industrial standards and simulations/experiments. We will therefore consider decreasing RTHT and keeping MTRT as in Eq. (2.3). Fig. 2.1 shows an example distribution of packet arrivals for a real-time channel within one MTRT (as in Eq. (2.3)). The horizontal axis

of this figure represents the number, $N$, of packet arrivals within one MTRT. If $N$ is a continuous (discrete) random variable, the vertical axis represents the probability density (mass) function of $N$. The shaded area right to $N_{max}$ represents the region where some packets of this real-time channel may be lost due to insufficient bandwidth reserved, where $N_{max}$ is the maximum number of packets this real-time channel can transmit during each token allocation. We will derive $N_{max}$ using the performance requirements such as Eq. (2.1) or Eq. (2.2), and Fig. 2.1. So, we can choose an RTHT large enough to satisfy this condition. That is, within this RTHT, the system must be able to transmit at least $N_{max}$ packets of this channel. $N_{max}$ can then be used to determine the RTHT directly by using the relationship $RTHT = N_{max} \times P_{max}$. There are many possible ways to derive the desired RTHT (or $N_{max}$) when $MTRT = D$, depending on the type of performance requirements. We consider the following three typical performance requirements. Let $G$ denote the *average* packet-arrival rate of a real-time channel.

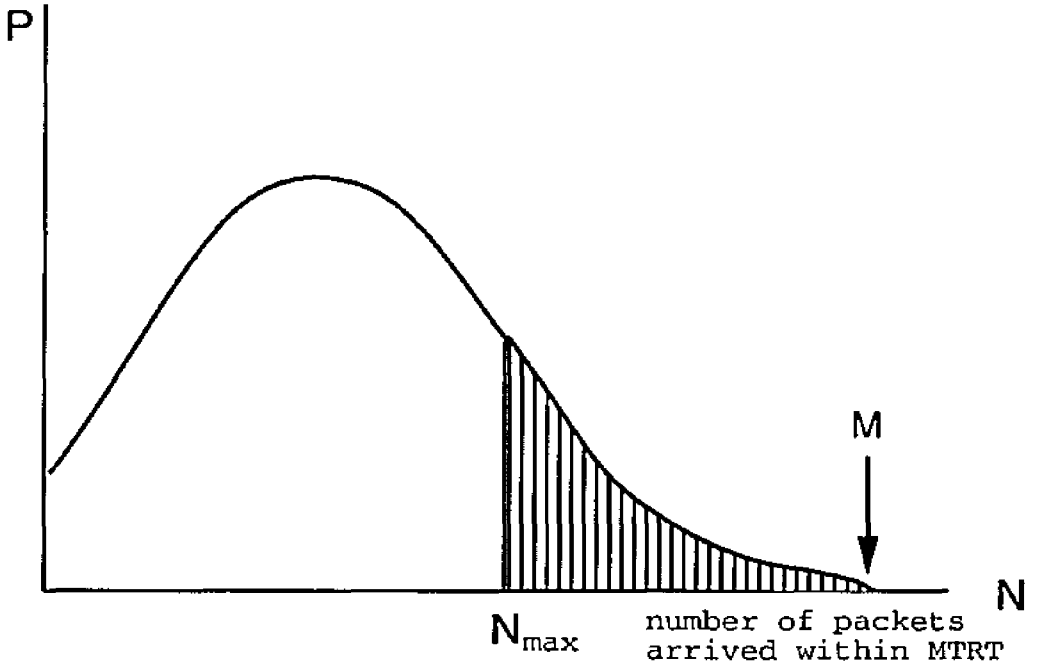



**Figure 2.1:** An example distribution of packet arrivals within one MTRT

1: *P( delay of a packet ≤ delay bound D ) ≥ a given number Z.*

This inequality is the same as Eq. (2.1) and should be satisfied over a sufficiently long time period. Considering Fig. 2.1, if the number of packet arrivals falls in the unshaded region to the left of $N_{max}$, all packets can be transmitted before their deadlines, since the node can transmit up to $N_{max}$ packets during each token allocation. Similarly, in

the shaded region right to $N_{max}$, at most $(n - N_{max})$ packets will miss their deadlines. Therefore, Eq. (2.1) can be converted to Eq. (2.6) by using Fig. 2.1. Without loss of generality, we can consider $N$ to be a discrete random variable, leading to:

$$Z \leq 1 - \frac{\sum_{n=N_{max}}^{M}(n - N_{max})P(n)}{\sum_{n=0}^{M} nP(n)} \tag{2.6}$$

$$= 1 - \frac{\sum_{n=N_{max}}^{M}(n - N_{max})P(n)}{G \times MTRT} \tag{2.7}$$

If $\sum_{n=N_{max}}^{M} P(n)$ is small, we can approximate Eq. (2.7) with

$$Z \leq \frac{\sum_{n=0}^{N_{max}} nP(n)}{G \times MTRT}. \tag{2.8}$$

Using either Eq. (2.7) or Eq. (2.8), we can find the smallest $N_{max}$ that satisfies this performance requirement.

With a minor modification, this case can be applied to many similar performance requirements, e.g., "messages" are considered instead of packets in the performance requirement formula. For example, if the performance requirement is

$$P(\textit{ delay of a message } \leq \textit{ delay bound } D) \geq \textit{ a given number } Z,$$

we can modify Eq. (2.7) to:

$$Z \leq 1 - \frac{\sum_{n=N_{max}}^{M} f(n - N_{max})P(n)}{G \times MTRT}, \tag{2.9}$$

where $f(n)$ is the number of messages which will miss deadlines given that $n$ packets will miss their deadlines.

2: *P( no packet loss during any time interval of length $\geq MTRT$ ) $\geq Z$.*

This is similar to Eq. (2.2). The unshaded area to the left of $N_{max}$ in Fig. 2.1 should be greater than, or equal to, $Z$. Therefore, the relation between $Z$ and $N_{max}$ is represented by:

$$Z \leq \sum_{n=0}^{N_{max}} P(n). \tag{2.10}$$

Eq. (2.10) can be used to find the smallest $N_{max}$ that makes the unshaded area to the left of $N_{max}$ greater than, or equal to, $Z$.

3: $P(\textit{delay of a packet} \leq \textit{delay bound } D) \geq \textit{a given } Z$ for all time intervals of length MTRT or larger.

This is the strictest requirement among the three cases, because it has to be satisfied during any time interval of length MTRT or larger. So, we must consider their worst case:

$$N_{max} \geq M \times Z. \tag{2.11}$$

We do not consider this performance requirement over an interval of smaller than MTRT, because in case of an interval smaller than MTRT, say, a very short period during which only one packet is lost, it is impossible to satisfy the requirement unless $Z = 0$, i.e., a hard real-time channel.

In Section 2.6, we will use an example of the first type of performance requirement and derive MTRT and RTHT with the proposed scheme.

## 2.5 Run-time Scheduling

Since each real-time channel can be described by its MTRT (equivalent to "period") and RTHT (equivalent to "running time"), we can use the deadline-driven scheduling algorithm in [8,29] for allocating tokens to individual real-time channels. Before the LCU grants a real-time channel establishment request, a new schedule for allocating tokens (e.g., "s0" in Fig. 2.2) must be computed in addition to an admission test. According to the schedule, the LCU issues a real-time token to each real-time channel at least once every MTRT, and this token allows a node, when it receives the token, to transmit packets of the corresponding channel for a period up to RTHT. In order to improve network utilization, the LCU tries to allocate a real-time token to each real-time channel exactly once every corresponding MTRT (if all channels use up all their reserved bandwidth) and use the remaining time for non-real-time traffic. When there are no scheduled activities in the schedule, the LCU issues a non-real-time token whose expiration time is set to the beginning of the next scheduled activity (including the token passing overhead), and this token circulates among all nodes on the network. Using an example we will illustrate these scheduling activities. In Fig. 2.2, suppose "s0" is the original schedule and time starts with 0. The symbol "ns" represents the time that can be used to schedule non-real-time traffic. If the token "A1" is returned at time $t$, the system will change the original schedule, s0, to a new schedule, s1, by moving

the rest of the entire schedule ahead by $(RTHT - t)$ time units. A1 (A2) in Fig. 2.2 is the first (second) token for channel A which is characterized by MTRT and RTHT.

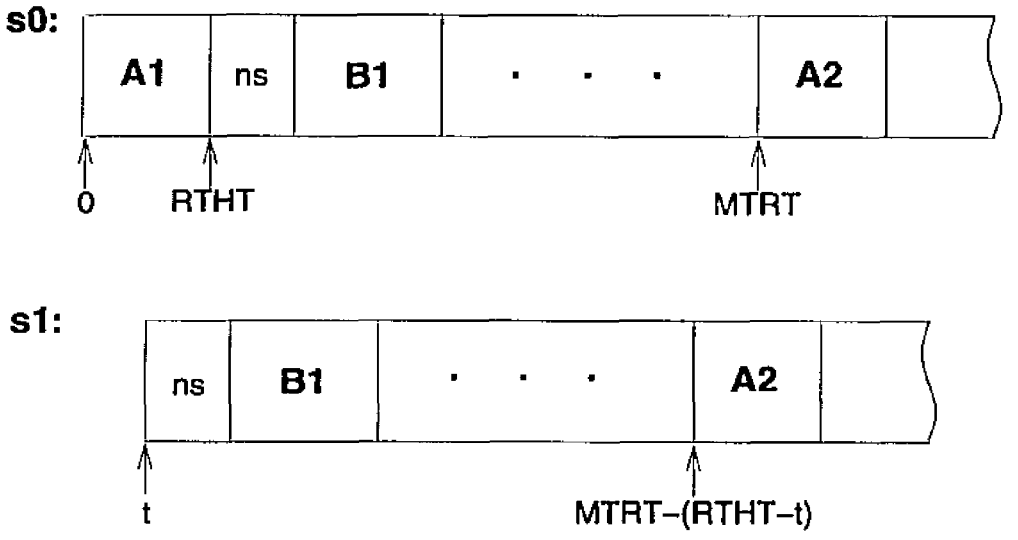



**Figure 2.2:** A run-time scheduling example

Runtime scheduling performed by individual nodes is simple. Each real-time token indicates which real-time channel should be scheduled, and each node discards late packets and transmits the remaining packets in the queue according to their deadlines. When a node receives a non-real-time token, it transmits non-real-time packets in the queue until the token expires, or continue to circulate the non-real-time token if the node completes the transmission of non-real-time packets before the token expires. Note that, in order to achieve a higher non-real-time traffic throughput, a node does not give real-time packets higher priority when it receives a non-real-time token.

Since no complex scheduling algorithm is required at the node level, the proposed scheme can be implemented on a very simple node which may not even have sufficient computing power, e.g., smart sensors in an automated factory. In the next section, we will show via simulation the effectiveness and efficiency of the proposed scheme.

The communication overhead for the proposed scheme is low, since the following two properties are generally true for typical (soft) real-time audio/video communication. The first is the time needed to establish a channel is usually much shorter than the life of an application. This overhead does not have significant effects on the overall communication cost, as this is only a one-time cost for a channel.

The second property is that the total size of successfully-delivered packets (per token allocation) of these applications is usually much larger than the size of a token. The ratio

of the average successfully-delivered traffic volume per token allocation to the size of two tokens (one to issue and the other to return) can be treated as the actual utilization of this real-time channel, since the token passing time is the primary source of communication overhead for our scheme. Although this ratio depends greatly on applications, it is usually small, because typical real-time audio/video applications generate large amounts of data. In the simulation example (Section 2.6), if we assume each token takes 500 bytes, the ratio is shown to be less 1% for the proposed scheme.

## 2.6 Simulation

In this section we present a numerical example to demonstrate the effectiveness of the proposed scheme. We will use both the FDDI and the proposed scheme to transmit compressed digital motion-video frames and compare their performances. This example shows that the proposed scheme reserves the bandwidth required for real-time traffic and also utilizes the network efficiently in the absence of real-time traffic.

### 2.6.1 Simulation Model

In order to make a fair comparison with FDDI, we simulated a 100 Mbps multiaccess link/bus with 20 and 50 nodes. The video data are sampled from a sequence of CNN headline news, stored on a laser disk [48]. The size of each frame, after JPEG compression [45,48], is plotted in Fig. 2.3. The quality of video can be characterized by the rate of 'successfully-delivered' frames, where a successfully-delivered frame is defined as one which is delivered to its destination correctly before the corresponding deadline. The maximum one-way transmission delay of each frame must be less than 100 ms in order to achieve the quality of live performance. If we use the transmission rate of 30 frames per second, 3 frames will be transmitted in each 100 ms. Assume the maximum packet size of the network is 1 Kbytes, and define the time to transmit a maximum-size packet as the *packet time*. Therefore, 100 ms is equal to 1250 packet times, or $D = 1250$ (in packet time). We will use a packet time as the basic time unit in the rest of this section. Following Eq. (2.3), we get $MTRT = D = 1250$ packet times. The performance requirement can then be specified as:

$$P(\textit{delay of a frame} \leq 1250) \geq \textit{a given } Z. \tag{2.12}$$

We will discuss three cases: $Z = 99\%$, 95% and 90%. (All other cases can be handled similarly.)

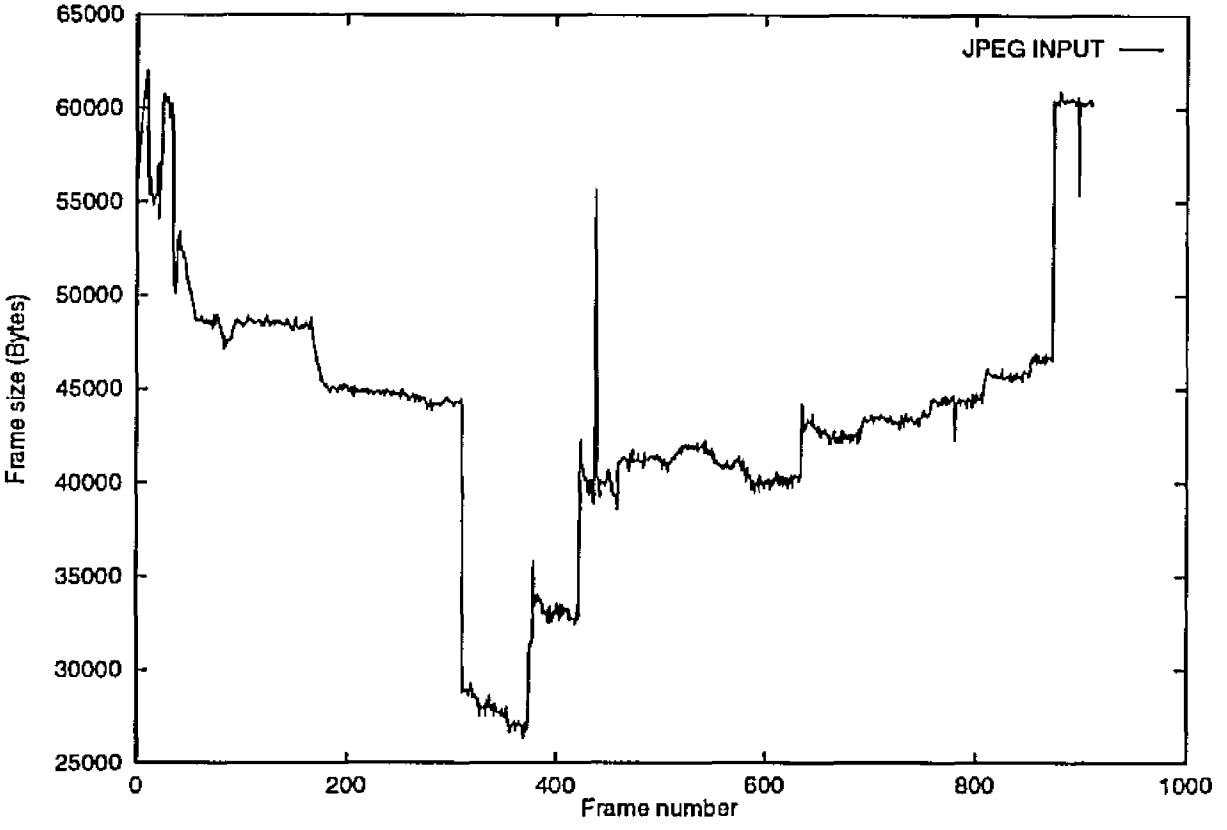



**Figure 2.3:** An example of frame sizes (JPEG compression)

We need the distribution of traffic arrivals within the time interval MTRT (=1250) to derive RTIIT. By adding the sizes of three consecutive frames (because there are three frame arrivals in 100 ms), we can derive the distribution of traffic volume (in Kbytes) within $D$ =1250. Fig. 2.4 shows the distribution of traffic volume (three frame arrivals) within one MTRT. For the scheme proposed in Section 2.4, we use this distribution to compute the bandwidth reservation for various frame-delivery rate requirements. Let $T_{max}$ be the time needed to transmit one maximum-size message. $T_{max}$ = 62 according to Fig. 2.3. The performance requirement can then be expressed as:

$$Z \leq 1 - \frac{\sum_{n=N_{max}}^{186} P(n)}{\sum_{n=80}^{186} 3P(n)} \tag{2.13}$$

$$= 1 - \sum_{n=N_{max}}^{186} \frac{1}{3} P(n), \tag{2.14}$$

since if $N_{max}$ is sufficiently large ($> M - T_{max} = 185 - 62 = 123$), each point in Fig. 2.4 will result in loss of at most one frame. This corresponds to $f(n) = 1$ in Eq. (2.9). Note that $\sum_{n=80}^{186} 3P(n)$ is the expected number of frames which will arrive within one MTRT (100 ms). By adding one packet time as the token passing overhead, we get:

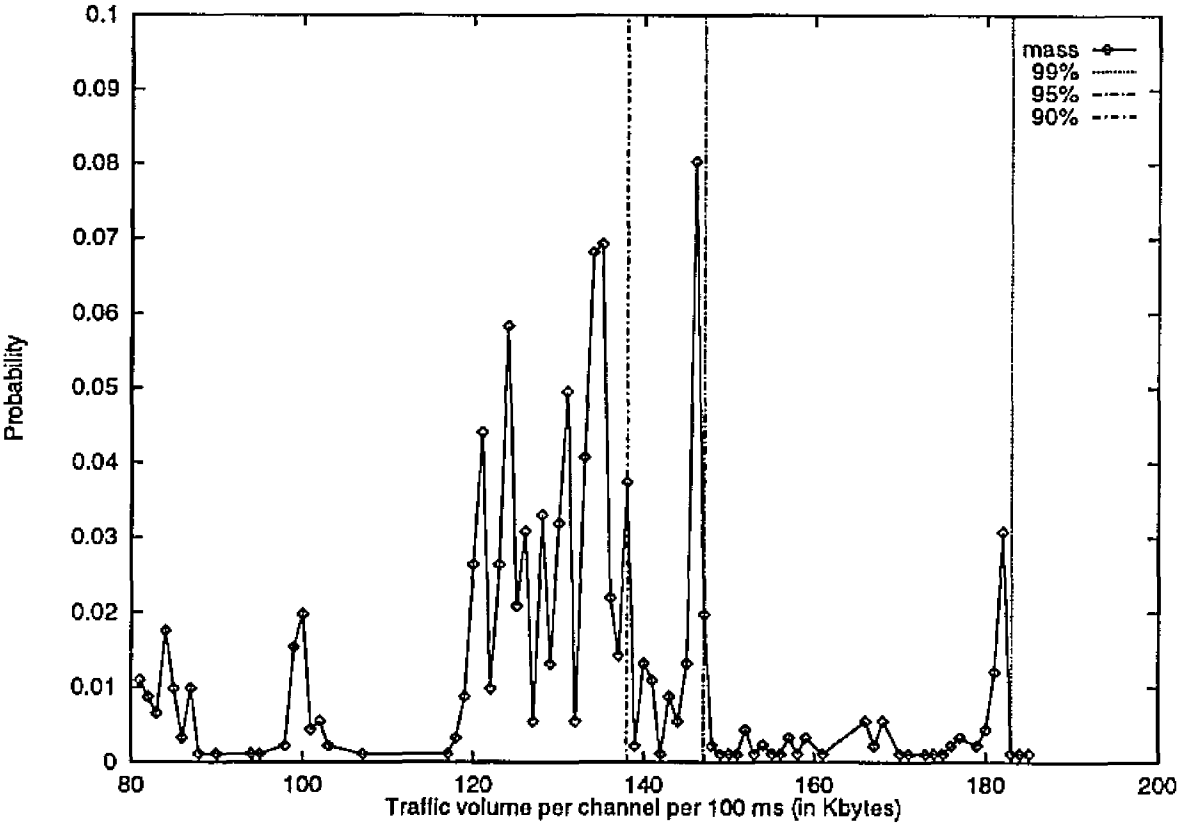



**Figure 2.4:** An example distribution of traffic volume in an interval of length 100 ms

- $Z = 99\%$: $N_{max} = 183$.
- $Z = 95\%$: $N_{max} = 147$.
- $Z = 90\%$: $N_{max} = 138$.

According to Eq. (2.5) the network is expected to support six 99% video channels, eight 95% video channels, or nine 90% video channels. Certain uniformly distributed non-real-time traffic that requires from 0% to 90% of the total link capacity is added to each node during the simulation. However, the source nodes of real-time channels are randomly chosen. (It is possible that one node may become the source node of all real-time channels.) The traffic data of real-time channels were taken from Fig. 2.3 and each channel has its own starting frame (also randomly chosen).

In both 20-node and 50-node FDDI rings, we use the same input traffic except we assume the source nodes of existing real-time channels are distributed evenly. The high-priority token holding time is also distributed evenly to all nodes on the FDDI ring. The token passing overhead is ignored in the FDDI case.

### 2.6.2 Simulation Results

The main goal of our simulation is to evaluate and compare the maximum and average *frame-miss rates* of both the FDDI and the proposed scheme. (The frame-miss rate is defined as the percentage of frames missing their deadlines.) We will also evaluate the improvement of network utilization by using statistical channels and examine the bandwidth available for the transmission of non-real-time traffic in the presence of real-time traffic. Our scheme is shown to always outperform FDDI and, at the same time, have the ability to provide performance guarantees.

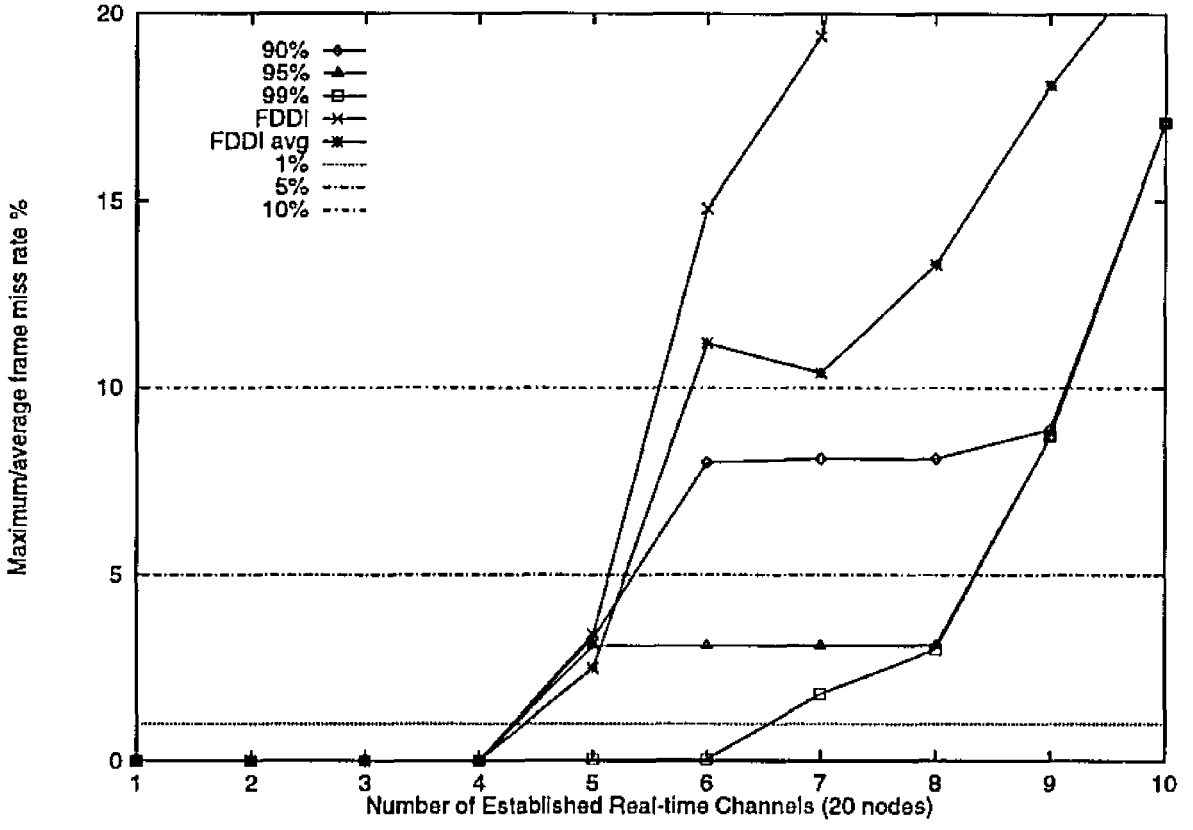



**Figure 2.5:** Maximum frame-miss rate at a 50% non-real-time traffic load (20 nodes)

Figs. 2.5 and 2.6 show the maximum frame-miss rate of our scheme (90%, 95%, and 99% lines), and the maximum and average frame-miss rates of 20-node and 50-node FDDI networks at a 50% non-real-time traffic load. The frame-miss rate is defined based on a channel, i.e., the ratio of the number of frame misses for a channel $C$ vs. the total number of frames of $C$. The maximum (average) frame-miss rate is defined to be the largest (average) value of individual channel frame-miss rates. Each point in the figure represents 30,000 cycles of the sequence for each channel, i.e., about 911,000 frames or 8.4 hours at the rate

of 30 frames per second for each channel. As predicted by the analytic model, both figures show that six 99%, eight 95% and nine 90% channels can be supported under the proposed scheme. For those points where the link does not have sufficient bandwidth for both real-time and non-real-time traffic, e.g., $7$–$10^{th}$ at 99% channels, $9$–$10^{th}$ at 95% channels and $10^{th}$ at 90% channel, we reserved the entire link capacity for real-time channels, and non-real-time traffic is transmitted only when the bandwidth reserved for real-time traffic is not used. Since we reserved the link bandwidth using the worst-case values, this is very likely to happen.

For example, we reserved $\lceil 1250/7 \rceil$ packet times for each channel when there are seven 99% real-time channels. (Note, however, that our scheme does not allow a $7^{th}$ 99% real-time channel, because the system cannot guarantee the required performance.) When a $7^{th}$ 99% channel is added, about 2% of real-time packets of this channel will miss deadlines. The FDDI can provide 4–5 real-time channels at a 50% non-real-time load. As we will see later, the FDDI's ability to support real-time communication is highly sensitive to the non-real-time traffic load. The average frame-miss rate for our scheme is very close to the maximum miss rate, so we only plot the maximum frame-miss rate in Figs. 2.5 and 2.6. By contrast, the average and maximum frame-miss rates of FDDI are significantly different when the FDDI ring cannot transmit all the real-time messages before their deadlines. The FDDI's frame-miss rate is also sensitive to the number of nodes on the ring, but our scheme can provide the *same* number of real-time channels regardless of the number of nodes on the multiaccess link. This observation implies that the variance of frame-miss rate of our scheme is very small, whereas the FDDI suffers a large variation of frame-miss rate.

Figs. 2.7 and 2.8 show the maximum frame-miss rate of our scheme (90%, 95%, and 99% lines) as well as the maximum (FDDI line) and average (FDDI avg line) frame-miss rates of 20-node and 50-node FDDI networks at a 90% non-real-time traffic load. At a high non-real-time traffic load like this, the FDDI becomes nearly incapable of supporting real-time communication. It can only support/handle two channels ($< 10\%$ frame-miss rate) in the 20-node ring and one channel in the 50-node ring. By contrast, our scheme is insensitive to the non-real-time traffic load. The system can still provide six 99%, eight 95% or nine 90% real-time channels at a high non-real-time traffic load. Again, since the average frame-miss rate of our scheme is very close to the maximum frame-miss rate, we plot only the maximum frame-miss rate. (Similarly to the previous case, the FDDI's average frame-miss rate significantly differs from its maximum frame-miss rate.) As far as the ability to support

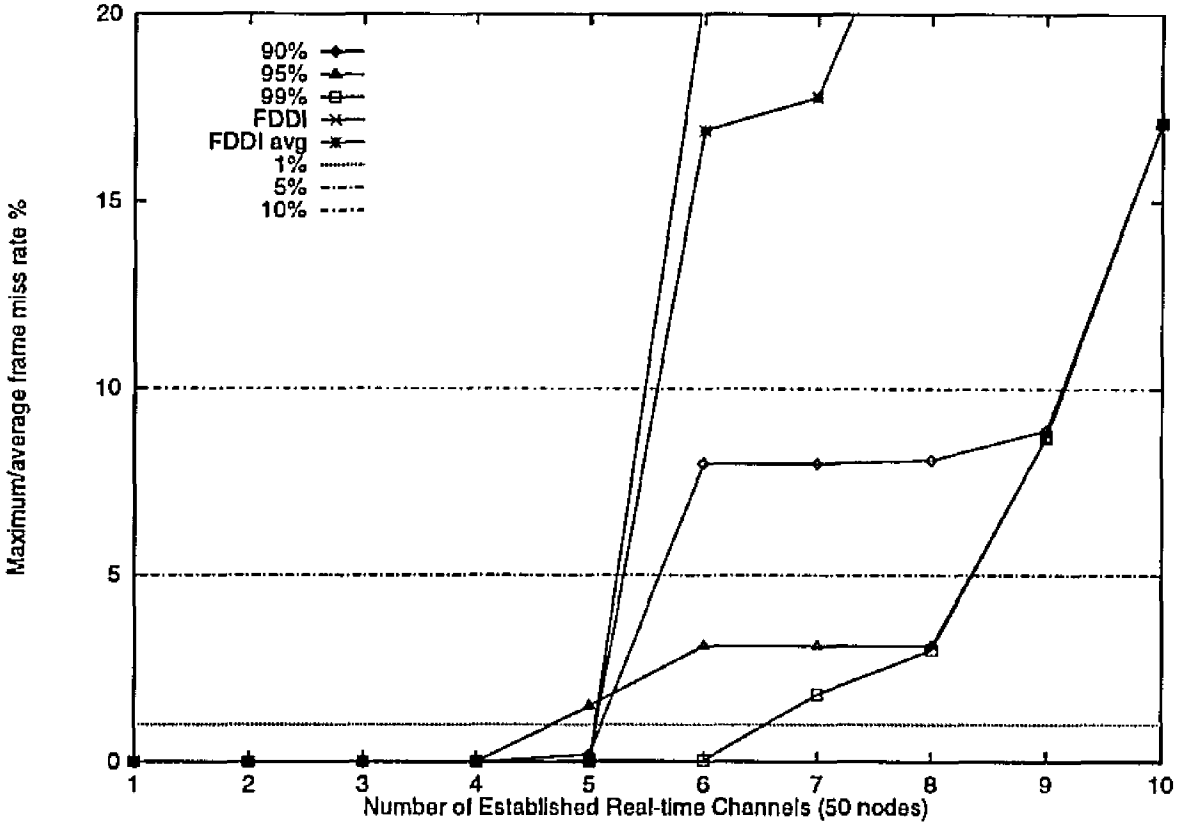



**Figure 2.6:** Maximum frame-miss rate at a 50% non-real-time traffic load (50 nodes).

real-time communication is concerned, our scheme is shown to be far better than the FDDI.

Our simulation also shows that network utilization can be improved significantly by using statistical channels. According to the simulation model, the network can support nine 90% channels, while only 6 channels can be established if no more than 1% of frames are allowed to miss their deadlines.

Fig. 2.9 shows the actual non-real-time traffic throughput (at a 90% non-real-time traffic load) and the capacity which is not reserved for real-time traffic under our scheme. As can be seen from this figure, the actual throughput is higher than the capacity which is not reserved for real-time traffic, because our scheme requires the token to be returned to the LCU if a node has no packets of the corresponding real-time channel to transmit. Thus, the unused portion of the reserved capacity can be used to transmit non-real-time packets, thus improving network utilization. This improvement is significant, especially when the reserved capacity is much higher than the average need. For example, when there are six 99% channels, only 12.2% of the link capacity is left for non-real-time traffic if we use circuit switching. In our scheme, however, the actual throughput is 36.5%, that is, it makes a 24.3% improvement over the circuit-switching case (in terms of total link capacity).

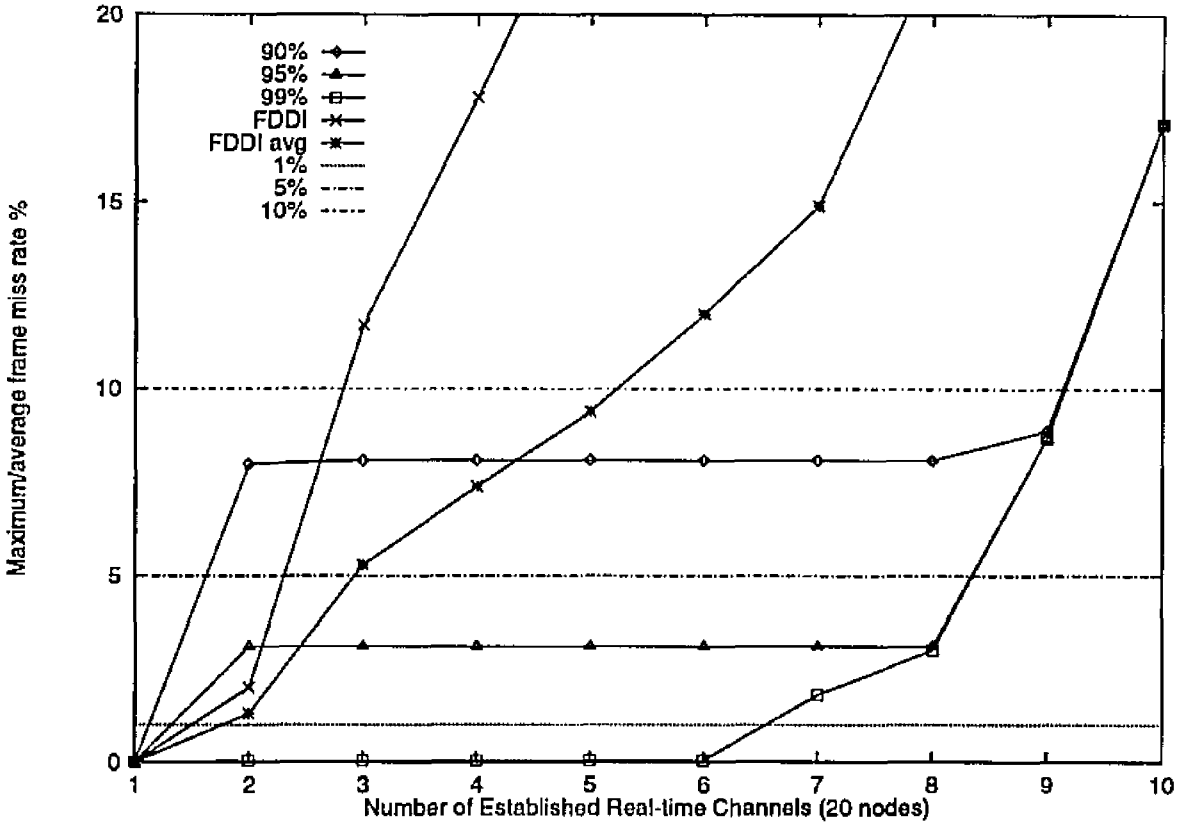



**Figure 2.7:** Maximum frame-miss rate at a 90% background load (20 nodes)

We also calculated the token passing overhead of the proposed scheme under the assumption that the size of a token is 500 bytes. Since two tokens are needed for each token allocation (one to issue and the other to return), the token passing overheads are approximately 0.84% (90% channels), 0.79% (95% channels), and 0.75% (99% channels), since the average successfully-delivered traffic volume per 100 ms are 119 Kbytes (90% channels), 127 Kbytes (95% channels), and 132 Kbytes (99% channels).

Since the average frame-miss rate and the maximum frame-miss rate of the proposed scheme are very close to each other, we can assume that whether a frame will miss its deadline or not (under the proposed scheme) follows a Bernoulli distribution and the number of missed/lost frames of each channel follows a Binomial distribution. Let $Y$ be a random variable denoting the number of lost frames of a channel. For a particular point in Figs. 2.5, 2.6, 2.7 and 2.8, let $n$ be the number of samples (frames) and $p$ be the frame-miss rate of a channel in the corresponding environment. Therefore, $n = 911,000$ and $p$ is the mean of $Y/n$. By applying the central limit theorem [20, 28],

$$\frac{Y - np}{\sqrt{n(Y/n)(1 - Y/n)}}$$

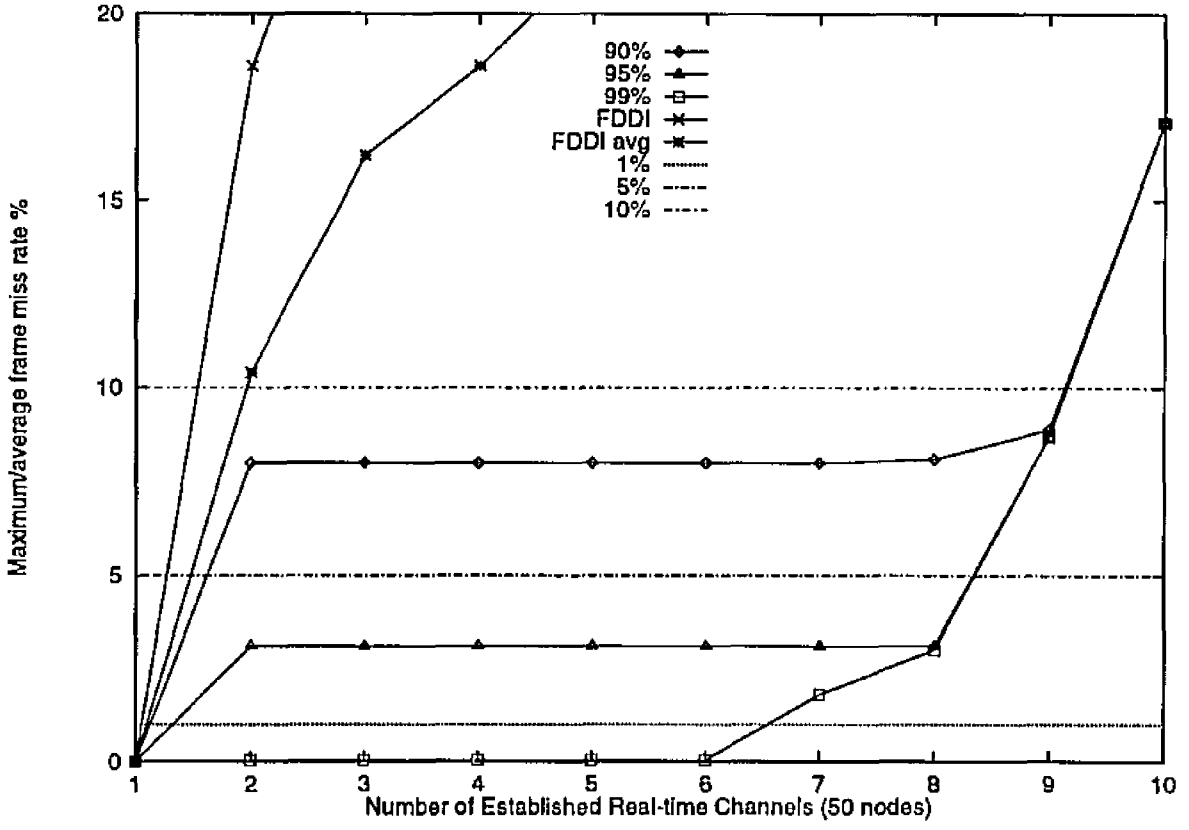



**Figure 2.8:** Maximum frame-miss rate at a 90% background load (50 nodes)

has a limiting distribution that is normal with mean 0 and variance 1. We can then find an approximate 99% confidence interval for the frame-miss rate, $p$, of our simulation. We can thus derive $P(-2.60 < \frac{Y-np}{\sqrt{n(Y/n)(1-Y/n)}} < 2.60) = 0.99$. As a result, for a large $n$, if the experimentally-determined value of $Y$ is $y$, then the interval

$$\left[ \frac{y}{n} - 2.60\sqrt{\frac{(y/n)(1-y/n)}{n}} \, , \, \frac{y}{n} + 2.60\sqrt{\frac{(y/n)(1-y/n)}{n}} \right]$$

provides an approximate 99% confidence interval of $p$. In our simulation ($n = 911,000$), for any experimental value $y$

$$2.60\sqrt{\frac{(y/n)(1-y/n)}{n}} < 0.003.$$

The 99% confidence interval is even smaller in cases where $k(> 1)$ channels exist, because $n = k \times 911,000$ in such cases, that is, the sample size increases.

Our scheme is shown to combine the advantages of circuit switching and packet switching. It provides performance guarantees for real-time channels and can also transmit non-real-time packets during the idle period reserved for real-time channels.

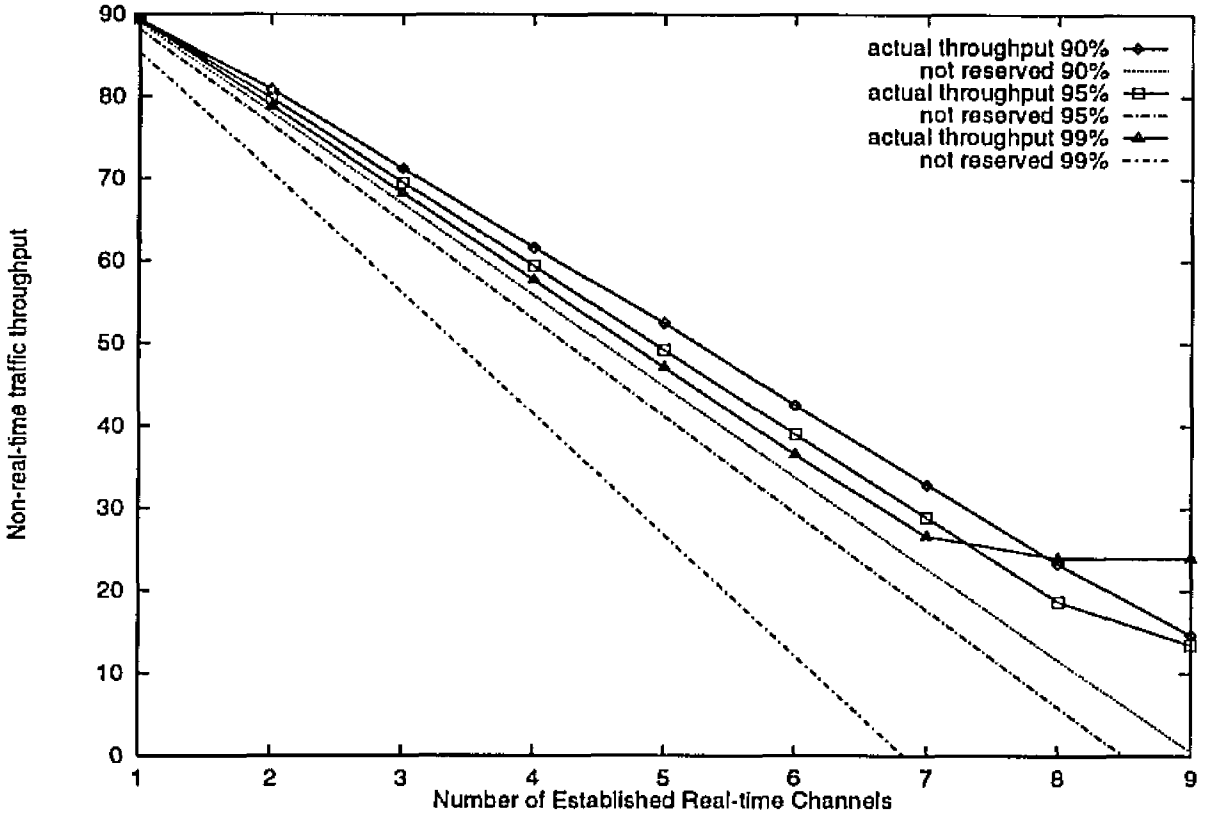



**Figure 2.9:** Non-real-time traffic throughput as % of total link capacity

## 2.7 Conclusion

In this chapter, we presented a new scheme for providing real-time performance guarantees given traffic-generation characteristics and performance requirements. In addition to its ability to provide performance guarantees, the proposed scheme can also improve network utilization by using statistical (as opposed to hard) real-time channels for the performance requirements specified in statistical terms. Since the traffic-generation model used in the proposed scheme is very general and the information needed for this scheme is easy to obtain, the scheme is easy to implement and is useful for many applications. Our simulation results have shown that this scheme is effective and efficient in supporting both real-time and non-real-time communication. In the next chapter, we introduce a channel-multiplexing strategy which further improves the network utilization and the ability to support real-time communication in a multiaccess network.

# CHAPTER 3

# MULTIPLEXING STATISTICAL REAL-TIME CHANNELS ON MULTIACCESS NETWORKS

## 3.1 Introduction

In Chapter 2, we proposed a scheme for real-time communication on multiaccess networks which can provide performance guarantees according to the user-specified traffic-generation characteristics and performance requirements. However, in order to let the system add/delete real-time channels independently, we used a *channel-based* design in Chapter 2 [13] (i.e., each channel was treated independently), but we did not consider the problem of *multiplexing* real-time channels. As a result, the channel-based scheme still under-utilizes the network. In this chapter, we significantly improve the channel-based scheme by multiplexing real-time channels originating from the same node in order to achieve higher network utilization without compromising the capability of independent addition/deletion and the performance guarantees of real-time communication.

Since more than one real-time channel may originate from a node, multiplexing these channels on a per-node basis may achieve higher network utilization and induce less overheads. That is, instead of reserving link bandwidth for each individual real-time channel, the system assigns a real-time token to a node at least once in a certain period of time for all real-time channels originating from that node. We may be able to multiplex several real-time channels (especially statistical real-time channels) with much less bandwidth than the sum of their individual bandwidths. However, the ability of independent addition and deletion of real-time channels must not be compromised while multiplexing real-time channels. We will therefore focus on the problem of making correct and efficient link-capacity reservation and run-time scheduling as well as preserving the ability of independent addition and deletion of channels.

In the rest of this chapter, we address the problem of multiplexing statistical real-time channels. Specifically, we propose (i) a scheme to calculate the link bandwidth associated with the addition or deletion of each channel, (ii) a procedure for establishing and closing a real-time channel, and (iii) a run-time scheduling algorithm for solving frame dependency problems. We decompose the channel multiplexing into three problems. The first problem deals with the issues associated with channel establishment. The second problem deals with the issues of closing a channel, and the third problem is concerned with the run-time scheduling.

The chapter is organized as follows. Our proposed solution to this problem is described in Section 3.2 and Section 3.3. In Section 3.4 we demonstrate via simulation the correctness and effectiveness of the proposed scheme. The chapter concludes with Section 3.5.

## 3.2 Node-Based Scheme

In this section, we focus on the problem of multiplexing *statistical* real-time channels. Note that we cannot multiplex two *hard* real-time channels without compromising their performance guarantees. However, we allow a hard real-time channel to be multiplexed with several statistical channels. In this case, this hard real-time channel is given the highest priority among all the multiplexed channels. In the rest of this section, the term, "real-time channel" or simply "channel", (unless stated otherwise) means a statistical real-time channel.

We propose an incremental scheme to add or delete a real-time channel and adjust the reserved link bandwidth when a new real-time channel is accepted or an existing channel is torn down. Real-time channels will be time-multiplexed on a per-node basis, since from a single node's point of view, once the node holds the token, it has the complete control of the link until its token-holding time expires. Note that the token-holding time here is referred to the real-time token-holding time for transmitting the packets of real-time channels. Basically, the proposed node-based scheme will function as follows. Each node $N$ will compute $MTRT_N$ and $RTHT_N$ for the combined traffic of all real-time channels which originate from node $N$. As will be seen later, $MTRT_N$ is the smallest MTRT of real-time channels which originate from node $N$. The LCU then allocates the real-time token to $N$ at least once every $MTRT_N$, which allows the node to transmit real-time packets up to $RTHT_N$ units of time. As in the channel-based token allocation in Chapter 2, the node

should return the token to the LCU immediately after it completes the transmission of all existing real-time packets or the token holding time expires, whichever occurs first. If the token for node $N$ is returned at time $t = 0$, then the next time node $N$ will receive the token is $t \leq MTRT_N - RTHT_N$.

We assume that the distribution of incoming traffic for all real-time channels is available at the time of receiving a channel-establishment request, i.e., information similar to Fig. 2.1 is available for each real-time channel. Although the exact distribution of incoming traffic may not be always available, an approximate distribution is usually not difficult to obtain off-line for most real-time applications. For example, during interactive playback of stored video, although the exact distribution depends on the user's on-line instructions, the original distribution from a sequential playback is usually a good approximation. For other applications like video conferencing, although we cannot predict in advance the exact distribution of the anticipated traffic, an approximation is usually not difficult to obtain, e.g., follow industrial standards, or run extensive simulations and make conservative estimations.

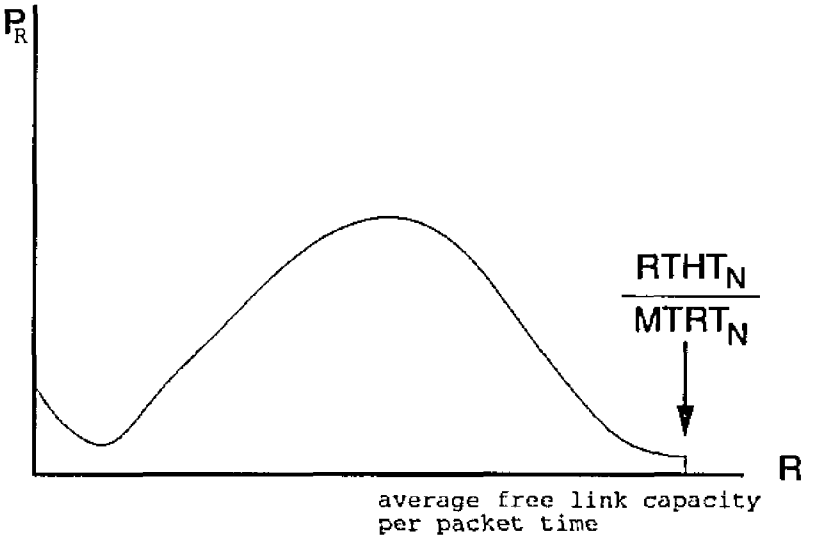



**Figure 3.1:** An example distribution of the RBU link bandwidth for a node in packet times

Before proceeding to the description of the channel-multiplexing scheme, we first introduce the distribution of *reserved-but-unused* (RBU) link bandwidth for a node, which is defined as the link bandwidth reserved by the node for real-time channels, but not actually used at run-time. This distribution will be used in the derivation of link bandwidth to be reserved when a new channel is to be added. Fig. 3.1 shows an example of the probability

distribution of a node's RBU link bandwidth per packet time measured in packet times (defined in Section 2.6). The vertical axis of Fig. 3.1 represents the probability mass of the RBU link bandwidth, and the horizontal axis represents the number of packets. The probability of no RBU link bandwidth is usually non-zero (i.e., $P$(RBU link bandwidth $= 0) > 0$), because the assigned capacity cannot otherwise be used up even in the worst case.

We will show how the distribution of the RBU link bandwidth (Fig. 3.1) is derived from the distribution of packet arrivals (Fig. 2.1) at the time of channel establishment or teardown and how it is used in the link-bandwidth reservation procedure. Basically, we want the new requesting channel to use as much RBU bandwidth as possible before requesting additional link bandwidth from the LCU.

### 3.2.1 Channel-Establishment Phase

**Establishing the first real-time channel of a node**

Since there is no real-time channel originating from the node, this request can be handled just as in the channel-based scheme. Hence, we can determine the MTRT and RTHT (or $N_{max}$) for this channel according to the incoming traffic characteristics (Fig. 2.1) and the performance requirement.

Because no real-time channel has already been established, there is no RBU link bandwidth for this node. The node sends the LCU this first request of real-time channel establishment which includes the MTRT and RTHT of this channel. If this request is admissible, the LCU will send a confirmation message to the requesting node. Then, the distribution of the RBU link bandwidth must be updated, since a new real-time channel has been accepted and the corresponding link bandwidth has been reserved. Before proceeding with the derivation of the RBU link bandwidth, we present two intuitive results in theorem form without proofs.

**Theorem 1** *Suppose $MTRT > 1$ (in packet time). If a real-time performance requirement can be satisfied by a real-time channel with the maximum token return time, $MTRT$, and a real-time token holding time, $RTHT$, then this performance requirement can also be satisfied by a pseudo real-time channel with link bandwidth $RTHT/MTRT$ per packet time.* □

**Theorem 2** *Suppose $MTRT > 1$ (in packet time). If a real-time performance requirement can be satisfied by a pseudo real-time channel with link bandwidth $RTHT/MTRT$ in*

*every packet time, and the delay bound of this channel's packets is at least $MTRT$, then this performance requirement can also be satisfied by any real-time channel with a maximum token return time $MTRT' \leq MTRT$ and a real-time token holding time $RTHT' \geq RTHT \times MTRT'/MTRT$.* □

Because different real-time channels may have different MTRTs, we have to use a common MTRT for the the distribution of RBU link bandwidth. The smallest possible MTRT is one packet time, and by Theorem 1, we can convert any real-time channel to a pseudo channel with link bandwidth $RTHT/MTRT$ in every packet time. By Theorem 2, we can also convert a pseudo real-time channel back to a real-time channel if the channel's MTRT is given. So, we choose one packet time as the basic time unit for the distribution of RBU bandwidth.

Let $X$ be the random variable representing the number of average packet arrivals for a new real-time channel within one packet time and $R$ be the random variable representing the RBU link bandwidth in a packet time. Let $N$ represent the number of packet arrivals within one MTRT, then

$$X = \frac{N}{MTRT} \tag{3.1}$$

$$R = \frac{RTHT}{MTRT} - X, \tag{3.2}$$

and,

$$\begin{aligned} p_R(r) &= P\left(\frac{RTHT}{MTRT} - X = r\right) \\ &= P\left(X = \frac{RTHT}{MTRT} - r\right), \text{ for } 0 < r < \frac{RTHT}{MTRT} \end{aligned} \tag{3.3}$$

$$p_R(0) = P\left(X \geq \frac{RTHT}{MTRT}\right), \tag{3.4}$$

where $p_R(r) = P(R = r)$ is the probability mass function of the RBU link bandwidth in a packet time.

We will use the notation $MTRT_n$ and $RTHT_n$ to denote as the maximum token return time and the real-time token holding time, respectively, for node $n$, while using $MTRT$ and $RTHT$ for a particular channel. If channels need to be distinguished, we will use $MTRT^n$ and $RTHT^n$ for channel $n$.

### A general procedure for channel establishment

A real-time channel-establishment request has to be handled differently than the previous case if it is not the first request. Basically, the procedure can be divided into two cases, depending on whether or not the current RBU link capacity of the node is sufficient to provide the required performance of this new channel without reserving any additional link bandwidth. If it is sufficient, the node can simply accept this channel without asking the LCU for more bandwidth. On the other hand, if the RBU link bandwidth of the node is not sufficient to meet the performance requirement of the newly-requested channel, the node has to determine the additional bandwidth needed for this new channel. Whether the current RBU link bandwidth is sufficient or not can be determined by the following three steps.

**Step 1:** Compute the distribution of $Y = R - X$ first, where $R$ is the current RBU link capacity in a packet time and $X$ is the average number of packet arrivals for this new channel within one packet time. Fig. 3.2 shows an example distribution of $Y$.

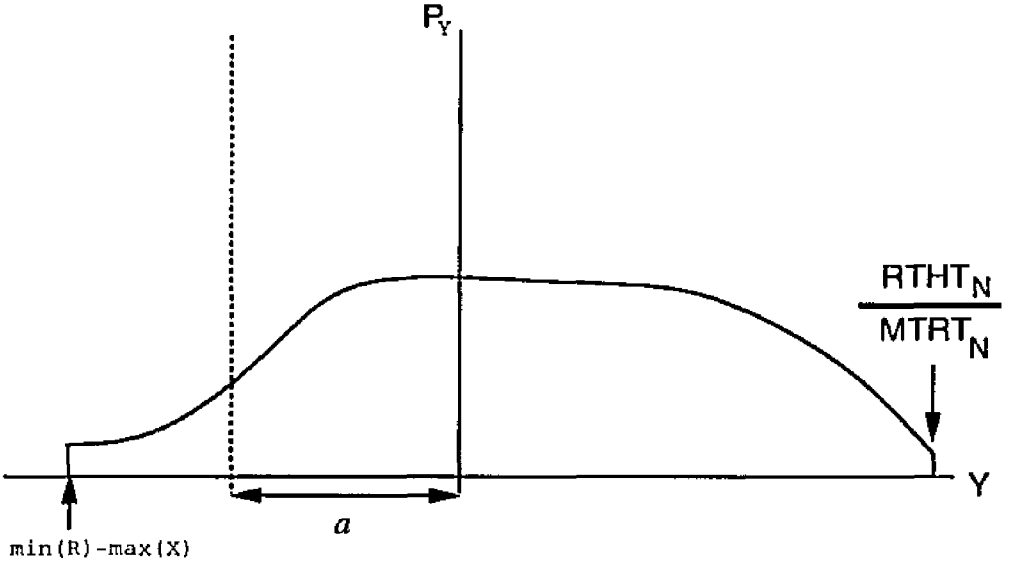



**Figure 3.2:** An example distribution of $Y = R - X$

$$
\begin{aligned}
p_Y(y) &= P(R - X = y) \\
&= \sum_x P(R = y + x | X = x) \times P(X = x) \qquad (3.5) \\
&= \sum_x P(R = y + x) \times P(X = x), \qquad (3.6)
\end{aligned}
$$

where $\min(R) - \max(X) \leq Y \leq \frac{RTHT_N}{MTRT_N}$, and $p_Y(y) = P(Y = y)$ is the probability mass function of $Y$. Because $R$ and $X$ are independent, Eq. (3.6) equals Eq. (3.5).

**Step 2:** The goal of this step is to determine the additional link capacity needed for this new channel in a packet time. As in Chapter 2, we use three typical performance requirements to illustrate our approach.

**P1:** *P(delay of a packet ≤ delay bound D) ≥ a given Z*: This requirement should be satisfied in an average sense, i.e., over a sufficiently long time period. Using Fig. 3.2 this performance requirement can be converted to:

$$Z \leq 1 - \frac{\sum_{n=\min(R)-\max(X)}^{-a}(-n-a)p_Y(n)}{G \times MTRT}. \tag{3.7}$$

Using Eq. (3.7), we can find the smallest $a$ for a given $Z$ and the distribution, as in Fig. 3.2. Since it is desirable that the reserved link bandwidth can be used up in the worst case, i.e., $\min(R) = 0$, we get

$$Z \leq 1 - \frac{\sum_{n=-\max(X)}^{-a}(-n-a)p_Y(n)}{G \times MTRT}. \tag{3.8}$$

If the requested channel is a hard real-time channel (i.e., $Z = 1$), we can derive $a = \max(X)$ from Eq. (3.8). In Eq. (3.15), this corresponds to reserving an additional bandwidth $a \times P_{max} \times MTRT_N$ for every $MTRT_N$. As discussed in Chapter 2, with a minor modification, this case can be applied to many similar performance requirements.

**P2:** *P(no packet loss during any time interval of length ≥ MTRT) ≥ Z*: The area left to the dotted line in Fig. 3.2 should be $\leq 1 - Z$. Thus, the relation between $Z$ and $a$ is represented as:

$$Z \leq 1 - P(Y + a \leq 0)$$

$$Z \leq 1 - \sum_{n=\min(R)-\max(X)}^{-a} p_Y(n) \tag{3.9}$$

$$Z \leq 1 - \sum_{n=-\max(X)}^{-a} p_Y(n). \tag{3.10}$$

Eq. (3.9) can be used to find the smallest $a$ that makes the area left to the dotted line not more than $1 - Z$. If $\min(R) = 0$ (normally), Eq. (3.10) can be used. Similarly, if a hard real-time channel is requested (i.e., $Z = 1$), $a = \max(X)$ can also be derived from Eq. (3.10).

**P3:** *P(delay of a packet ≤ delay bound D) ≥ a given Z* for all time intervals of length ≥ MTRT. This is a more strict requirement than **P1** and **P2**, because it has to be satisfied during any time interval of length not less than MTRT. So, we must consider its worst case:

$$Z \leq 1 - \frac{|\min(R) - \max(X) + a|}{\max(X)} \text{ or,} \tag{3.11}$$

$$Z \leq 1 - \frac{|-\max(X) + a|}{\max(X)}, \text{ if } \min(R) = 0. \tag{3.12}$$

As in **P1** and **P2**, we can find the smallest $a$ satisfying Eq. (3.11) or Eq. (3.12), and $a = \max(X)$ can be derived if the requested channel is a hard real-time channel.

**Step 3:** The $a$ derived from **P1–P3** represents the additional link capacity in one packet time needed to accommodate this new channel without compromising the quality guarantees of existing channels. Obviously, the case of $a \leq 0$ represents the current RBU link capacity of node $N$ is sufficient to provide the required performance of the new channel, and thus, node $N$ can accept this new channel without asking the LCU for more bandwidth, if $MTRT^{new\ channel} \geq MTRT_N$. After accepting this new channel, the RBU link capacity of node $N$ has to be updated as:

$$p_{R_{new}}(0) = P(Y \leq 0) \tag{3.13}$$

$$p_{R_{new}}(n) = p_Y(n),\ \forall n > 0. \tag{3.14}$$

If the MTRT needed for this new channel is smaller than the current $MTRT_N$, node $N$ still has to ask the LCU for a smaller $MTRT_N$ with a message containing the new $RTHT_N$ and $MTRT_N$ for this node:

$$MTRT_{N,new} = MTRT^{new\ channel}$$

$$RTHT_{N,new} = RTHT_{N,old} \times \frac{MTRT_{N,new}}{MTRT_{N,old}}$$

The probability that the LCU will grant this request (for the new $MTRT_N$) is high, because the request does not increase the bandwidth that needs to be reserved. However, as $MTRT_N$ decreases, the corresponding token passing overhead increases, and thus, this request may not always be accepted.

If the current RBU link capacity of node $N$ is not enough to guarantee the required performance of this new channel (i.e., the $a$ derived in Step 2 is greater than zero), then

we first determine the new $MTRT_N$. If the MTRT of the new channel is smaller than the current $MTRT_N$, then let $MTRT_{N,new} = MTRT^{new\ channel}$; otherwise, $MTRT_N$ remains unchanged.

After determining $MTRT_N$, we can compute the new $RTHT_N$ as:

$$RTHT_{N,new} = RTHT_{N,old} \times \frac{MTRT_{N,new}}{MTRT_{N,old}} + a \times MTRT_{N,new} \times P_{max}. \quad (3.15)$$

In Eq. (3.15), $a \times MTRT_{N,new} \times P_{max}$ represents the additional link capacity needed in one $MTRT_{N,new}$. Note that if $a = \max(X)$ (requesting a hard real-time channel to be established), $\max(X) \times MTRT_{N,new} \times P_{max}$ is equal to the link capacity necessary to establish a hard real-time channel in the channel-based scheme, because

$$\max(X) \times MTRT_{N,new} = \max(N) \times \frac{MTRT_{N,new}}{MTRT^{new\ channel}}.$$

Node $N$ sends the LCU a channel-establishment request message which contains the new $MTRT_N$ and $RTHT_N$. The LCU will try to reserve the requested bandwidth and reply with either accept or reject. If the reply from the LCU is reject, the new channel request cannot be accepted, so $MTRT_N$, $RTHT_N$ and the distribution of RBU link capacity remain unchanged. On the other hand, if the reply is accept, the RBU link capacity of node $N$ has to be updated as:

$$p_{R_{new}}(0) = P(Y + a \leq 0) \quad (3.16)$$

$$p_{R_{new}}(n) = p_Y(n - a),\ \forall n > 0. \quad (3.17)$$

### 3.2.2 Channel-Deletion Phase

Deletion of an existing real-time channel must also be done independently of other existing channels. After closing an existing real-time channel, we must update the RBU link capacity using the performance requirement and the distribution of the traffic arrivals of the deleted channel. Basically, the capacity reserved for the deleted channel is added to the current RBU link capacity of the node from which the deleted channel originates. If there is an excessive RBU link capacity, we may try to decrease the link capacity to be reserved for the node and/or return the excessive capacity to the LCU.

An intuitive way to achieve the above goal is to develop a procedure which can "add" the bandwidth used by the deleted channel back to the current RBU link capacity of this node without compromising the performance guarantees of other real-time channels. Let $X$ and $R$ be defined as in Eqs. (3.1) and (3.2), except that $X$ now represents the number

of packet arrival of the deleted channel. Therefore, the new RBU link capacity can be expressed as $R_{new} = R_{old} + X$. However, since $X$ and $R_{old}$ are not independent, we cannot easily compute the distribution of $R_{new}$ from the distributions of $R_{old}$ and $X$.

In order to derive the distribution of $R_{new}$, we need to re-compute MTRT and RTHT for the remaining channels as if we were to try to re-establish them in the original order of their arrival. If there is excessive RBU link capacity, i.e., $p_R(0) = 0$, the node returns it to the LCU to make $p_R(0) > 0$.

This method will reserve only the necessary link capacity, as it is basically the re-computation of link bandwidths for adding channels. However, if the number of remaining real-time channels is very large, its computation may become too expensive to be practical. Instead of re-computing the required link capacity reservation, we introduce an alternative, using the Fourier Transform to directly add the capacity of the deleted channel back to the RBU link capacity based on the performance requirement and the traffic-arrival distribution of the deleted channel. However, since the re-computation is in fact much simpler than the computation of Fourier Transform, unless the number of channels is very large, we will usually re-compute the link capacity to be reserved for the remaining channels. The efficiency of handling a channel-closing request is not as important as that of handling a new channel-establishment request, because the system can always close the channel first and compute the RBU link bandwidth later.

As defined in the previous section, let $R$ represent the current RBU link capacity, and $N$ represent the number of packet arrivals within one MTRT of the closing channel. According to the performance requirements and traffic-generation characteristics of this closing channel, we can derive $N_{max}$ for this closing channel. Since the system will schedule packets for no longer than $N_{max}$ packet times for this closing channel in one MTRT (see the run-time scheduling in the next subsection), we modify the probability distribution of packet arrivals as:

$$p_{N'}(N_{max}) = P(N \geq N_{max}) \tag{3.18}$$

$$p_{N'}(n) = p_N(n), \ \forall n < N_{max}. \tag{3.19}$$

Let $X' = \frac{N'}{MTRT}$ and let $K$ denote the total reserved link capacity (in one packet time) for all the real-time channels originating from this node at a given instant. Let $U = K - R$, which is a random variable representing the portion of reserved link capacity which was actually used during run-time. Since $K$ is a constant, the distribution of $U$ can be easily

obtained from the distribution of $R$. Let $X_1', X_2', \ldots, X_n'$ be the numbers of packet arrivals of $n$ existing channels (including the closing channel), then $U \geq \sum_{i=1}^{n} X_i'$, where $X_i' = \frac{N_i'}{MTRT}$ and $N_i'$ is defined as in Eqs. (3.18) and (3.19). Since $P(N \geq N_{max})$ is small in most applications, $U \approx \sum_{i=1}^{n} X_i'$. Thus, the distribution of $U$ is approximately the convolution of all distributions of $X_i'$'s, $i = 1, \ldots, n$, i.e., $\mathcal{F}(U) \approx \prod_{i=1}^{n} \mathcal{F}(X_i')$. Let $V = U - X'$ then $\mathcal{F}(V) = \frac{\mathcal{F}(U)}{\mathcal{F}(X')}$ and the distribution of $V$ can be obtained from the inverse Fourier Transform of $\mathcal{F}(V)$. The new RBU link capacity can also be approximated by $R_{new} = K - V$. Since $R_{new}$ is an approximation of the available RBU capacity and the RBU link capacity will be used when adding future channels, $R_{new}$ must not be larger than the actually available RBU. Because $U \geq \sum_{i=1}^{n} X_i'$, $V$ is larger than, or equal to, the portion of reserved link capacity which is not RBU after closing the channel. Therefore, $R_{new}$ is less than, or equal to, the actually available RBU, i.e., it is safe to use $R_{new}$ as an approximation of RBU.

## 3.3 Multiple-Due-Date Scheduling Algorithm

As in the channel-based scheme, the deadline-driven (normal Earliest Due-Date first) scheduling algorithm [29] can be used by the LCU (node-based scheme) to schedule tokens for real-time channels. When the total utilization (including overhead) is less than 1, the deadline-driven scheduling algorithm can guarantee all deadlines as long as the input traffic follows the pre-specified traffic-generation characteristics.

For individual nodes, the scheduling algorithm has to consider the existence of message (frame) inter-dependency. If all messages (frames) are independent of each other in a data stream, i.e., the performance requirement of a real-time channel can be characterized directly by the delivery rate of messages (frames) of the channel, then the deadline-driven scheduling algorithm can also be used as the primary scheduling discipline by each individual node. Since only $N_{max}$ packet times are reserved in one MTRT for a channel, the system has to give lower priority to the channel which had transmitted at least $N_{max}$ packets during the previous MTRT. This strategy can prevent the burstiness of some channels from degrading other channels' performance. Special customized scheduling policies can also be added easily at the channel-level, i.e., a node can give a certain channel higher priority according to the requirements of the application at hand.

If there exists inter-dependency among the messages (frames) of a data stream, i.e., the "effective" delivery of some frame depends on the delivery of some other frame(s), the

normal EDD scheduling algorithm alone is not sufficient to provide adequate performance guarantees. In this section, we will focus on the problem of scheduling real-time data streams in the presence of frame inter-dependency and propose a multiple-due-date (MDD) scheduling policy [14] to solve the problem.

### 3.3.1 Run-time Scheduling

Before proceeding to the description of the MDD scheduling algorithm, we first present an example of the frame-dependency problem. Many real-time applications, such as audio/video applications, often generate high-volume traffic. Hence, their real-time frames are usually compressed before transmission through the network. For example, MPEG uses three kinds of compressed frames: Intra-picture (I) frames, Predicted (P) frames and Interpolated (B) frames. I frames are coded directly from a picture and thus can be used to reconstruct a picture independently. P frames are coded with reference to a past picture (I or P). B frames are coded with reference to past or future pictures (I or P). It is obvious that a P or B frame will not be useful if the referenced frame(s) is not available. Therefore, the frame "delivery" rate is not an adequate measure of quality of MPEG-coded video. Instead, the frame "reconstruction" rate is a more appropriate measure of video quality.

In order to solve the frame inter-dependency problem, we need a scheduling method for systematically dropping/re-ordering packets in the outgoing queues so as to meet the performance requirement in the form of Eq. (3.21). Assume frame A is encoded only with reference to all of the frames in a set $S$. We want to ensure that frame A will not be transmitted until all frames in $S$ are transmitted. In this way, the performance requirement in the form of Eq. (3.21) will be directly implied by the performance requirement in the form of Eq. (3.20). Note that we assume that each data stream is independent of other data streams, and frame-dependency occurs only between frames within the same data stream.

$$P(\textit{delay of a frame} \leq \textit{delay bound } D) \geq \textit{a given } Z. \tag{3.20}$$

$$P(\textit{a frame can be reconstructed by its deadline}) \geq \textit{a given } Z. \tag{3.21}$$

The frame inter-dependency problem can be divided into two subproblems. First, we need an algorithm to compute the appropriate amount of link capacity to be reserved when a channel establishment request is received. The node-based scheme described in Section 3.2

can efficiently reserve sufficient link bandwidth for streams without frame inter-dependency. With a minor modification, the node-based scheme can also be used for real-time streams with frame inter-dependency. We will discuss the necessary modification later.

We need to ensure that a frame which already missed its deadline will not be discarded if it is still useful for the reconstruction of future frame(s). That is, no frame in $S$ will be discarded if frame A is not discarded. In order to solve this problem, we associate a frame with two due dates which are used in the MDD scheduling algorithm:

- *Scheduling-due-date* is used in the normal EDD scheduling algorithm. This is used to determine the transmission order of frames.
- *Drop-due-date* indicates the time this frame is no longer useful. This is used to determine the time that the frame will be discarded, if it has not already been transmitted completely.

Initially, the scheduling-due-date of a frame will be set to the earliest time the frame will be used for reconstructing some frame (not necessarily itself) at the receiver node. This time is often called the "delivery deadline" of the frame. Essentially, MDD works exactly the same as EDD except that MDD systematically changes the scheduling-due-date of a frame when necessary. By changing the scheduling-due-date of a frame, we also change the delivery order and the priority of this frame. The drop-due-date of a frame will be set to the time when the frame will be no longer useful for frame reconstruction at the receiver node.

After these two due dates of a frame are set, the frame enters the outgoing queue which will be scheduled for transmission by the normal EDD algorithm. Note that frames in the outgoing queue are in the ascending order of their scheduling-due-dates. When a frame reaches the head of the outgoing queue, there are two possible ways to handle this frame. If it can be transmitted completely before its scheduling-due-date, we start to transmit this frame immediately. Otherwise MDD will check whether this frame will be useful in the future or not. If this frame is not useful after its current scheduling-due-date, i.e., the transmission cannot be finished before its drop-due-date, it will be discarded. On the other hand, if this frame will be useful some time later, say $t$, we will set its scheduling-due-date to $t$, where $t$ must be greater than its current scheduling-due-date and less than or equal to its drop-due-date. After the new scheduling-due-date is determined, this frame will be inserted back into the outgoing queue right after all frames with smaller scheduling-due-dates.

We then need a systematic method to determine the next scheduling-due-date (i.e., $t$) for a frame which misses its current scheduling-due-date but will still be useful in the future. We propose to attach a list of "next" scheduling-due-dates to each frame. Each next scheduling-due-date represents a time when this frame will be used. The list is arranged in ascending order and each next scheduling-due-date is a value relative to the generation time of the frame. The last entry of the list will be the drop-due-date of the frame, and thus, can be omitted. If a frame is not referenced by other frames, the drop-due-date will be equal to the initial value of the scheduling-due-date and the entire list can be omitted. Since the loss of an "important" frame will lead to the loss of all frames which are coded with reference to it, the number of dependent frames is usually small to prevent a visible blackout.

If the input stream is highly periodic, the list of next scheduling-due-dates can be further simplified or omitted. For example, video frames are usually generated at a constant rate and expected to be reconstructed at the same rate by the receiver node, i.e., the inter-arrival time between frames is a constant. Therefore, the immediate next scheduling-due-date can be computed by adding the inter-arrival time to the current scheduling-due-date until the drop-due-date is reached. So, MDD is particularly useful for the transmission of video frames.

### 3.3.2 Link Capacity Reservation

If there is no frame inter-dependency, the node-based scheme (with EDD) in Section 3.2 can reserve link capacity efficiently according to the given performance requirement and the distribution of packet arrivals. However, due to the frame inter-dependency, we have to use MDD instead of the normal EDD for scheduling in order to meet the performance requirement in the form of Eq. (3.21). The adoption of MDD also implies that the amount of data "expected" to be scheduled for transmission does not follow the given arrival distribution when some "important" frame misses its initial scheduling-due-date. That is, the given distribution of packet arrivals is not the traffic distribution in the outgoing queue when important frames miss their scheduling-due-dates. We will use an example to illustrate the problem and then propose a solution for the general case.

**Example:** Assume a stream of MPEG-coded video frames (at the rate of 30 frames per second) has one I-frame and seven P-frames for every eight frames [19,30]. I-frames are

coded independently and P-frames are coded with reference to the past I-frame. Therefore, the previous I-frame is necessary for the reconstruction of P-frames. Note that this example is also used in our simulation.

Under MDD, since I-frames will not be discarded until their drop-due-dates and we always try to send I-frames before their associated P-frames, the delivery of P-frames will be delayed if the previous I-frame misses its scheduling-due-date. This will affect the amount of data that needs to be delivered for the reconstruction of the P-frames associated with the missed I-frames. Let $P$ be the probability that an I-frame will miss its scheduling due-date. The amount of data "expected" to be scheduled for the transmission and reconstruction of the next P-frame before the next scheduling-due-date (i.e., 33.3 ms later after the I-frame's initial scheduling-due-date) is the size of the P-frame plus $P \times S_I$, where $S_I$ is the size of an I-frame. Similarly, the amount of data expected to be scheduled for the reconstruction of the $n$-th next P-frame before the $n$-th future scheduling-due-date (i.e., $n \times 33.3$ ms later) is the size of the P-frame plus $P^n \times S_I$, where $n \leq 7$ in this case. Since $P$ is usually small (e.g., less than 0.1), this effect diminishes rapidly, i.e., the modified distribution is usually quite similar to the originally given one but shifted to right by a few packets.

**General Case:** Let $F$ and $R_i, 1 \leq i \leq n$, be frames in a stream carried by a real-time channel with frame reconstruction rate $P$ under the MDD scheduling. Let $S(R_i)$ denote the size of frame $R_i$. Assume $F$ is coded only with reference to $R_1 \ldots R_n$, and the initial scheduling-due-date of $F$ corresponds to the $m_i$-th next scheduling-due-date of $R_i$. Then under MDD, the amount of data that is expected to be transmitted for the reconstruction of $F$ is

$$S(F) + \sum_{i=1}^{n} P^{m_i} \times S(R_i). \tag{3.22}$$

Using Eq. (3.22), we can compute the expected size of each frame and also the distribution of traffic expected to be scheduled. Having the modified distribution of expected packet arrivals, we can use it as the given distribution to make link capacity reservation in the node-based scheme (with MDD).

## 3.4 Verification and Evaluation

We present in this section a numerical example to demonstrate the effectiveness of the MDD scheduling algorithm and the channel-multiplexing strategy. The transmission of

compressed digital motion-video frames with both channel-based and node-based schemes is used to show the significant improvement in network utilization as a result of channel multiplexing. Our simulation results also show that the channel-multiplexing scheme can still reserve a sufficient link capacity for real-time traffic to provide the guaranteed performance.

### 3.4.1 Simulation Model

We use a 100 Mbps multiaccess link/bus as the physical medium for transmitting digital video frames. The video data used are obtained from a 5376-frame (about 3 minutes at the rate of 30 frames per second) sequence of the movie "Star Wars" [30]. The size of each frame, after MPEG compression [19,30], is plotted in Fig. 3.3.

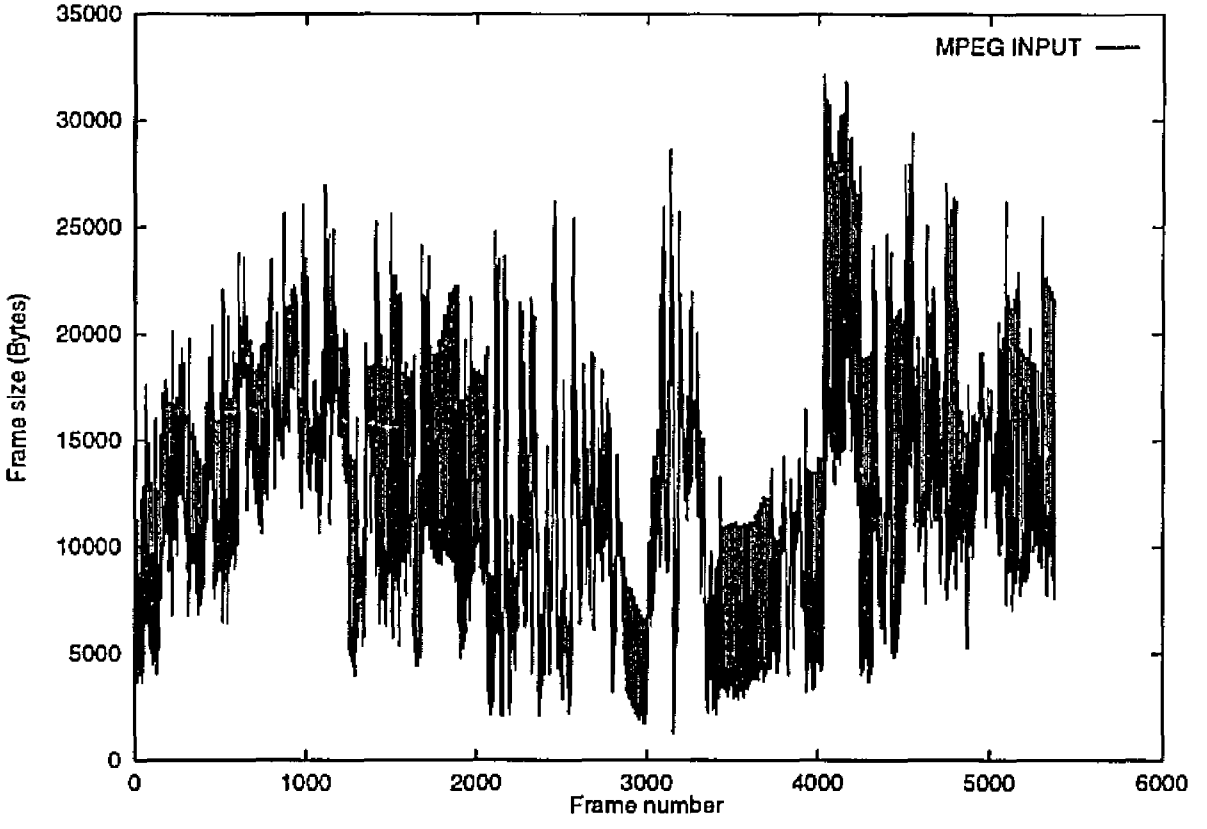



**Figure 3.3:** An example of frame arrivals

The maximum one-way transmission delay of a frame is assumed to be less than 100 ms in order to achieve the quality of live video. Note that the transmission delay (100 ms) includes only the queueing delay and the actual transmission time, i.e., encoding, decoding and other processing times are not included. At the transmission rate of 30 frames per second, 3 frames will be transmitted in each 100 ms. Although the original data rate was

24 frames per second, we use the typical live-video speed, 30 frames per second, in our simulation, because we want to simulate the requirement of live performance like video-conferencing. In order to make the simulation more realistic, we allow random jitters in the frame arrivals of each channel. These jitters are assumed to be uniformly distributed between [−416,417] (packet times). Note that 416.7 is the frame inter-arrival time. The maximum packet size of the network is assumed to be 1 Kbyte, so 100 ms is equal to 1250 packet times or $D = 1250$. The packet time will be used as the basic time unit in the rest of this section. The performance requirement for these video frames is assumed to be

$$P(a\ frame\ can\ be\ reconstructed\ by\ its\ deadline) \geq a\ given\ Z. \tag{3.23}$$

In our simulation, the MPEG video compression algorithm generates one I-frame and seven P-frames for every eight frames. Since the I-frames are coded independently, they can be used to reconstruct a picture independently. The P-frames are coded with reference to the previous I-frame, so the previous I-frame is necessary for the reconstruction of P-frames. Since the compression introduces the frame-dependency between P-frames and its previous I-frame, the delivery rate of frames does not directly imply the same frame reconstruction rate. That is,

$$P(delay\ of\ a\ frame \leq 1250) \geq a\ given\ Z \tag{3.24}$$

does not imply Eq. (3.23).

By using the proposed MDD scheduling algorithm, if a frame misses its scheduling-due-date, the system will reset the scheduling-due-date to some time later when this frame will be used again (before its drop-due-date). Thus, if an I-frame arrives at time $t$, its scheduling-due-date will be set to $t + 1250$ (in packet time) and its drop-due-date will be set to $t + 1250 \times 8/3$. Every time an I-frame misses its scheduling-due-date, the scheduling-due-date will be extended by $1250 \times 1/3$ packet times until its drop-due-date is reached. Similarly, if a P-frame arrives at time $t$, both of its due-dates will be set to $t + 1250$, since a P-frame has no value after its scheduling-due-date.

Since the maximum one-way transmission delay is 100 ms and the performance requirement is given in statistical form, by Eq. (2.3), $MTRT = 1250$. We need the distribution of traffic arrivals within one MTRT to derive RTHT, the link capacity to be reserved. By adding three consecutive frame sizes, we can derive the distribution of traffic arrivals (in Kbytes) within one MTRT. Fig. 3.4 shows the distribution of traffic arrivals within one MPEG channel.

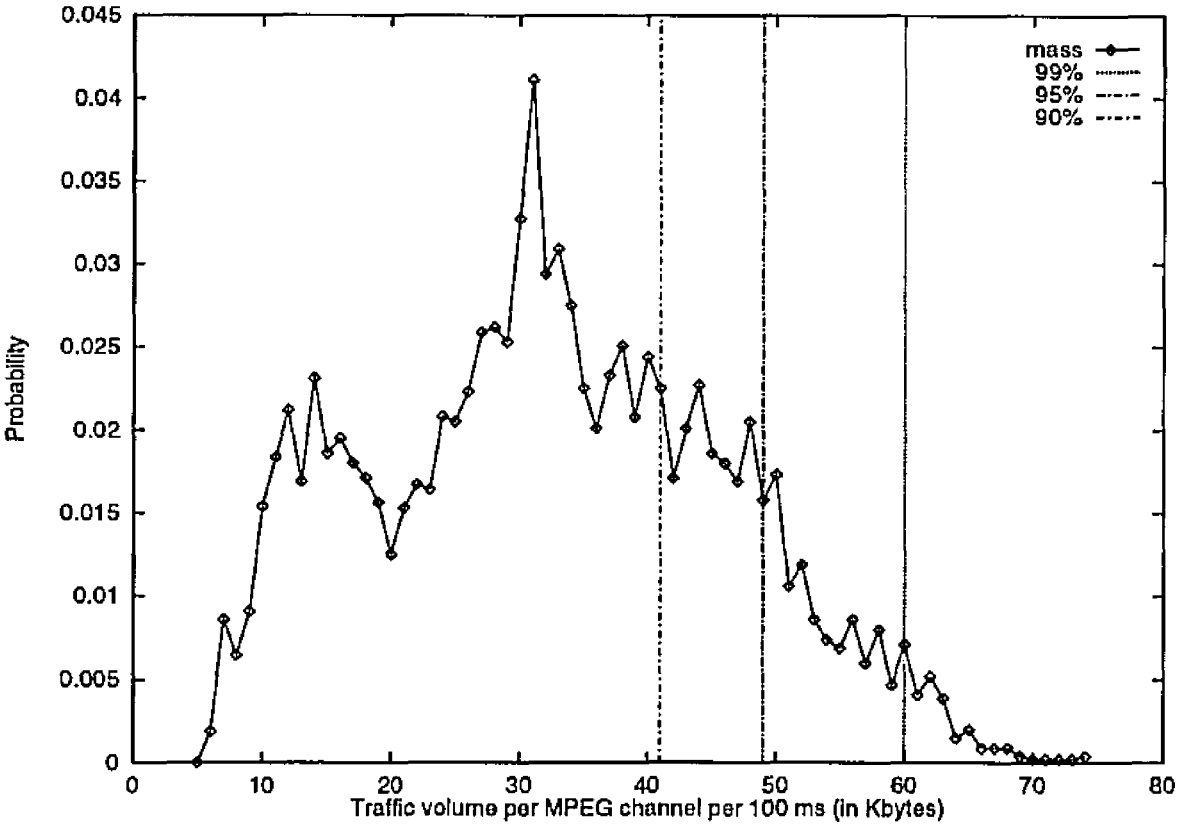



**Figure 3.4:** An example distribution of MPEG frame arrivals

The performance requirement can then be expressed as

$$Z \leq 1 - \frac{\sum_{n=N_{max}}^{74} P(n)}{\sum_{n=6}^{74} 3P(n)} = 1 - \sum_{n=N_{max}}^{74} \frac{1}{3} P(n), \tag{3.25}$$

since if $N_{max}$ is sufficiently large, each point to the right of the corresponding dotted line in Fig. 3.4 will result in loss of at most one frame. By adding 0.5 packet time per frame as the operation overhead, we get:

- $Z = 99\%$: $N_{max} = 60$.
- $Z = 95\%$: $N_{max} = 49$.
- $Z = 90\%$: $N_{max} = 41$.

Then we apply the algorithm proposed in Section 3.3 to derive the modified distribution according to the given frame rate. Fig. 3.5 shows the modified distribution of the "expected" traffic arrivals of one (90%) MPEG channel, i.e., 10% miss rate under the MDD scheduling policy. The performance requirement can then be expressed as:

$$Z \leq 1 - \frac{\sum_{n=N_{max}}^{77} P(n)}{\sum_{n=6}^{77} 3P(n)} = 1 - \sum_{n=N_{max}}^{77} \frac{1}{3} P(n), \tag{3.26}$$

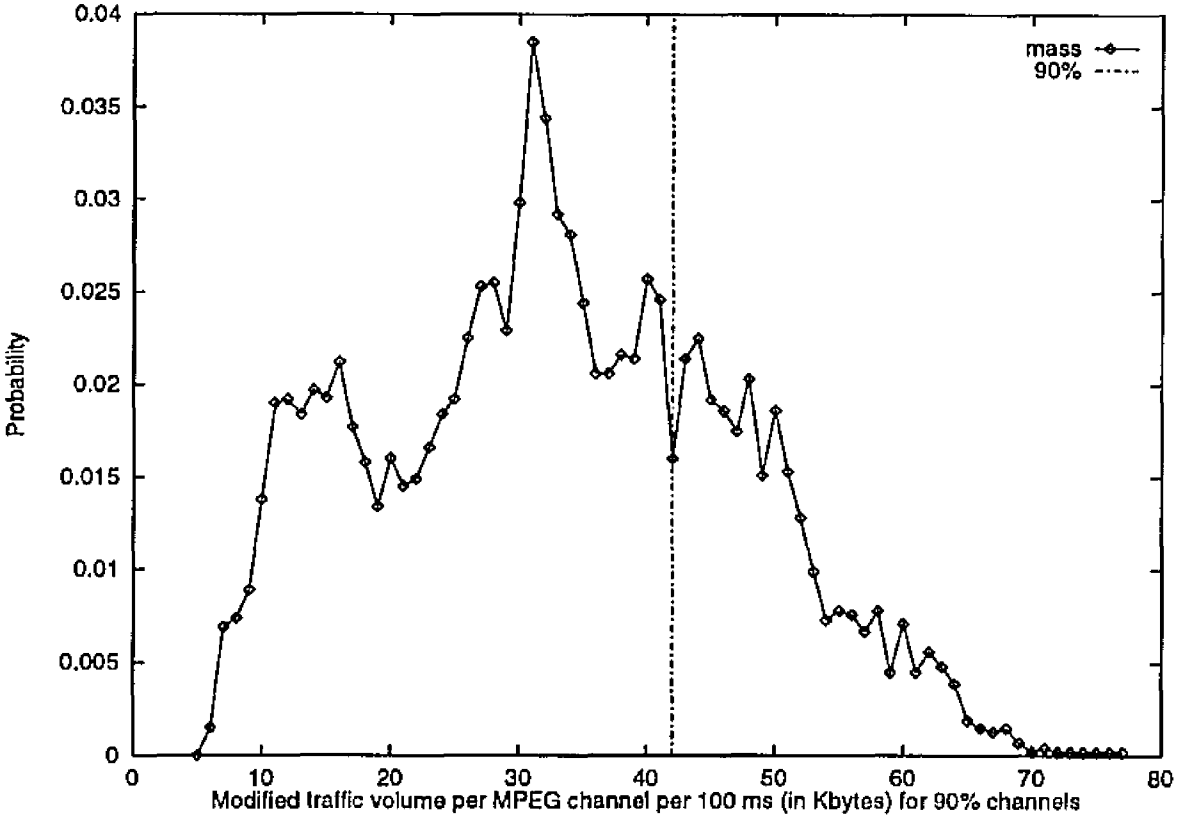



**Figure 3.5:** An example distribution of modified MPEG frame arrivals

Again, if $N_{max}$ is sufficiently large, each point to the right of the dotted line with label 90% in Fig. 3.5 will result in loss of at most one frame. By adding 0.5 packet time per frame as the operation overhead, we get:

$Z = 90\%$, (i.e.,$P = 0.1$): $N_{max} = 42$.

The subsequent link capacity to be reserved for new channels is also based on this modified distribution of traffic arrivals when the MDD scheduling policy is used. Using the same method, we can also obtain new modified distributions of frame arrivals for 95% and 99% channels. However, since the miss rate is small in these two cases, $N_{max}$ does not change for both 95% and 99% channels. Therefore, without channel multiplexing, the network is expected to support twenty-one 99% MPEG channels, twenty-six 95% MPEG channels, or thirty 90% MPEG channels, according to Eq. (2.5). As we shall see later, the node-based multiplexing scheme reduces significantly the total link capacity that needs to be reserved, and our simulation results confirm its effectiveness.

### 3.4.2 Simulation Results

The goal of our simulation is to demonstrate the effectiveness of the channel-multiplexing strategy and the MDD scheduling algorithm and show that the integrated scheme can provide performance guarantees even in the presence of frame-dependency.

By using the node-based scheme in Section 3.2 and the modification algorithm proposed in Section 3.3, the additional link capacity for a new MPEG channel can be determined based on the number of already-established real-time channels. Fig. 3.6 shows the normalized link capacity needed to add a new (node-based) channel with respect to the average traffic arrival rate of the channel.

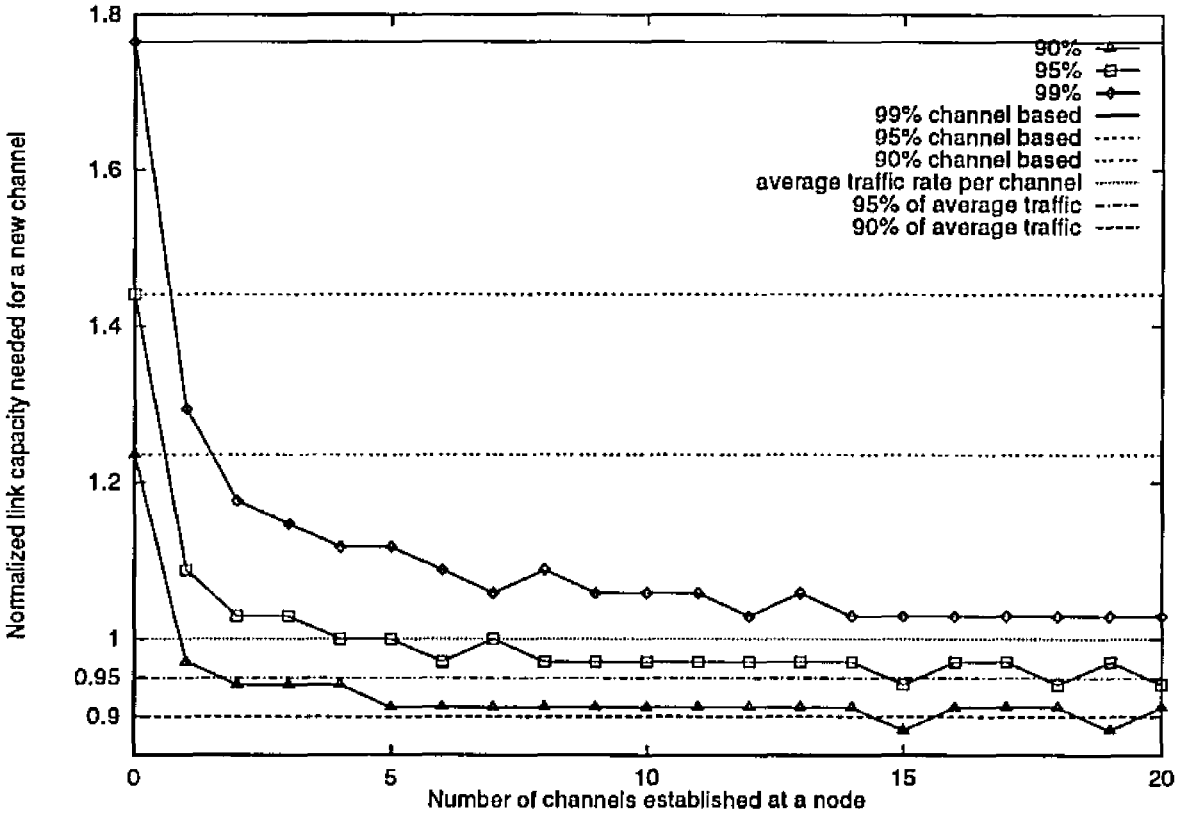



**Figure 3.6:** Normalized link capacity needed for adding a new channel with respect to the average traffic rate per channel

In Fig. 3.6, the horizontal lines "99% channel based", "95% channel based" and "90% channel based" correspond to the link capacity needed for a new 99%, 95% and 90% channel, respectively, if the channel-based scheme is used. The channel-based scheme requires a constant link capacity for adding a new channel regardless of the number of already-established

channels on that node, i.e., it represents the amount of reserved capacity needed to guarantee the performance if no multiplexing occurs. By contrast, in the channel-multiplexing scheme, although the first (node-based) channel still requires to reserve the same link capacity, it requires much less to add a new channel when there is already at least one established channel.

The capacity needed for adding a new 90% channel rapidly converges to the dotted line "$y = 0.9$" representing the average real-time traffic which needs to be delivered timely to achieve the required 90% frame reconstruction rate. That is, approximately 90% of the packets of a channel has to be delivered by their deadlines. Thus, the system only needs to reserve the link capacity according to about 90% of the *average* real-time traffic rate of a new channel when there are sufficiently many channels originating from a node (about 5 in this example). In this case, only 73% capacity of a 90% channel-based channel is needed. Thus, the network is expected to support about forty 90% channels, as compared to 30 without channel multiplexing.

The 95% (and 99%) line also demonstrates the same trend as the 90% line, i.e., converges to a constant which is close to 95% (and 99%) of the average real-time traffic arrival rate of a channel. Note that the line "$y = 1$" can be considered as the line "$y = 0.99$" here. As the required delivery rate increases, channel multiplexing becomes more effective. For 95% channels, 95% of average arrival traffic corresponds to only 67% of the capacity reserved for a 95% channel-based channel. For 99% channels, only 58% capacity of a 99% channel-based channel is needed. Thus, the network is expected to support about thirty-four 99% and thirty-seven 95% channels, as compared to 21 and 26 without channel multiplexing. Therefore, the network utilization and real-time channel admissibility are improved significantly with channel multiplexing. In other words, if there are sufficiently many channels originating from a node, we can provide performance guarantees for statistical real-time channels by reserving the link capacity based on the *average* case rather than the worst case.

After the additional capacity for a new channel is determined, we can then demonstrate the effectiveness of the proposed MDD scheduling algorithm. We use the frame-reconstruction miss rate in Figs. 3.7–3.10 of real-time channels to show that MDD is effective and works correctly. Note that the frame-reconstruction miss rate is defined as the percentage of frames which can not be reconstructed by their deadlines at the receiver node.

We will first present the simulation results for channels with a short lifetime (3 minutes to 1 hour). Although the statistical guarantees are defined based on the assumption of

infinitely long arrival data streams, as can be seen later, most (more than 99%) of channels with a short lifetime can still provide the required performance. Note that we use 99-percentile instead of the maximum miss rate in the short-lifetime channel experiments, since the maximum is too sensitive to a single random sample and does not reflect the real distribution of the data. However, we will use the maximum miss rate in the long-lifetime simulations to show our scheme can provide the promised performance guarantees.

Figs. 3.7–3.9 show the frame-reconstruction miss rates among all established channels under both the MDD scheduling and the normal EDD scheduling with the proposed channel multiplexing scheme, i.e., the link-bandwidth reservation is made based on the data in Fig. 3.6. Each unit of lifetime represents 5374 frames, i.e., about 3 minutes. The "20" represents all channels with lifetime 20 or more. Each line in these three figures represents a certain percentile (or average) among all samples with the same lifetime. For example, in Fig. 3.7, the line "MDD:90%" shows the 90-percentile frame-reconstruction miss rate of all 99% real-time channels with the same lifetime under MDD scheduling, and the line "EDD:average" in Fig. 3.8 shows the average frame-reconstruction miss rate of all 95% real-time channels with the same lifetime under the normal EDD scheduling.

The MDD scheduling performs significantly better than the normal EDD in terms of frame-reconstruction rate. As can be seen in Figs. 3.7–3.9, the 99-percentile frame-reconstruction miss lines of MDD scheduling are all below the corresponding required miss rate lines, and thus, can provide the required performance guarantees.

The performance of EDD is sensitive to the performance requirement. For 99% channels (Fig. 3.7), although EDD scheduling is still outperformed by the MDD scheduling, the 99-percentile miss rate line is below the 1% miss rate line, i.e., at least 99% of channels can meet the performance requirements. However, the 99-percentile miss rate line of 95% channels (Fig. 3.8), lies around "$y = 0.05$" and the 99-percentile miss rate line of 90% channels (Fig. 3.9) lies around "$y = 0.13$". In fact, even the average miss rate line of EDD lies above "$y = 0.1$" in Fig. 3.9. Thus, EDD scheduling is not appropriate for the short lifetime streams of non-independent frames.

The capability of providing performance guarantees can be shown in the long lifetime channel simulations. Fig. 3.10 shows the frame-reconstruction miss rate for multiplexing long lifetime channels. As can be seen from the figure, MDD (with the node-based scheme) works very well, i.e., the maximum frame-reconstruction miss rate among all channels is always kept under the corresponding required upper bound before the network is saturated.

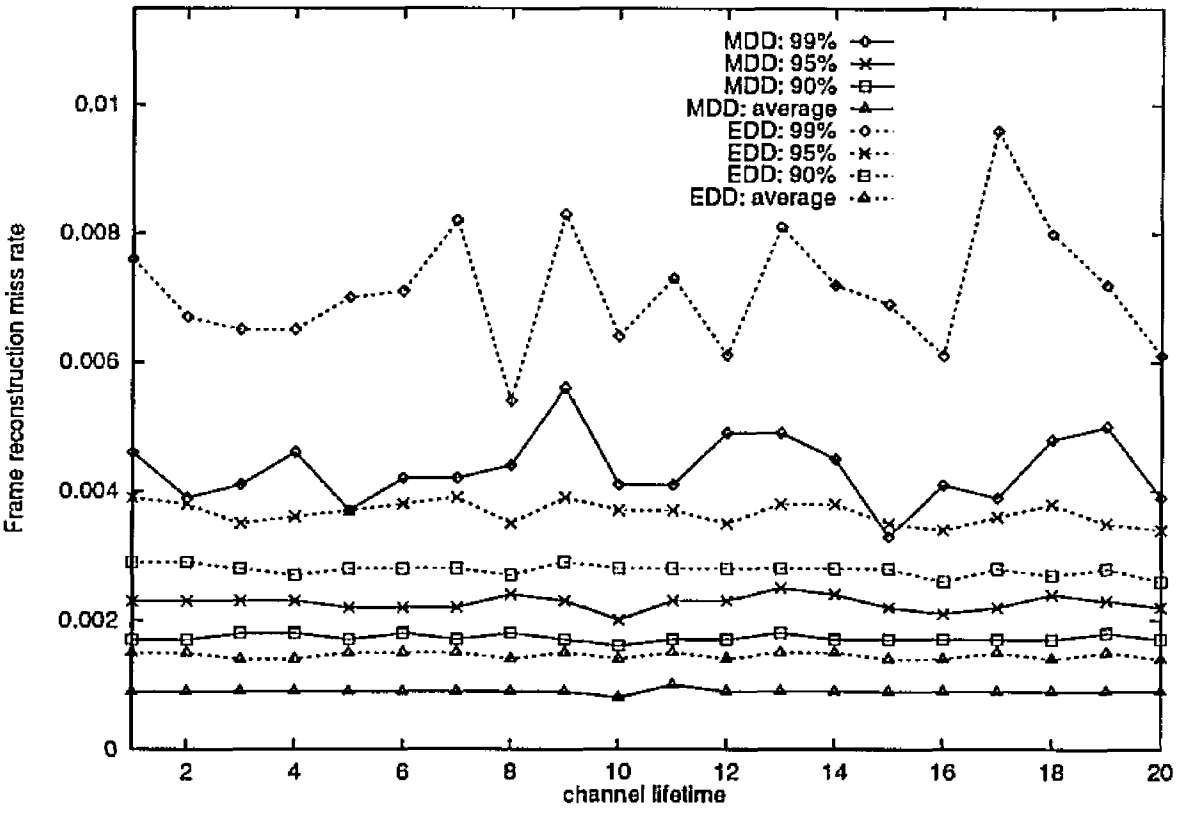



**Figure 3.7:** Frame-reconstruction miss rate for multiplexing 99% channels (short lifetime)

Each point in these figures represents the transmission of about 24,000,000 frames per channel or 222 hours at the rate of 30 frames per second. From Fig. 3.10, the network with MDD (and the node-based scheme) can provide performance guarantees for up to thirty-four 99% channels, thirty-seven 95% channels, or forty 90% channels, as expected from the capacity reservation schemes proposed in Section 3.2 and 3.3. In Fig. 3.10, the frame-miss rate starts high for the first channel and drops to the lowest point when there are about five 95% or 90% channels, then rises very slowly until the network is saturated. This trend can be explained by the reserved link capacity for adding a new channel. For the first 5 channels, we reserve more than the average need for each channel, so the frame-miss rate keeps dropping. After that, the capacity we reserve for a new channel is approximately equal to the average need of each channel. Thus, the (approximately) same amount of the reserved-but-unused capacity is shared by more and more channels so that the frame-miss rate rises slowly. The miss rate of 99% channel is actually dropping until the network is saturated, because the system always reserves more than the average traffic of a channel.

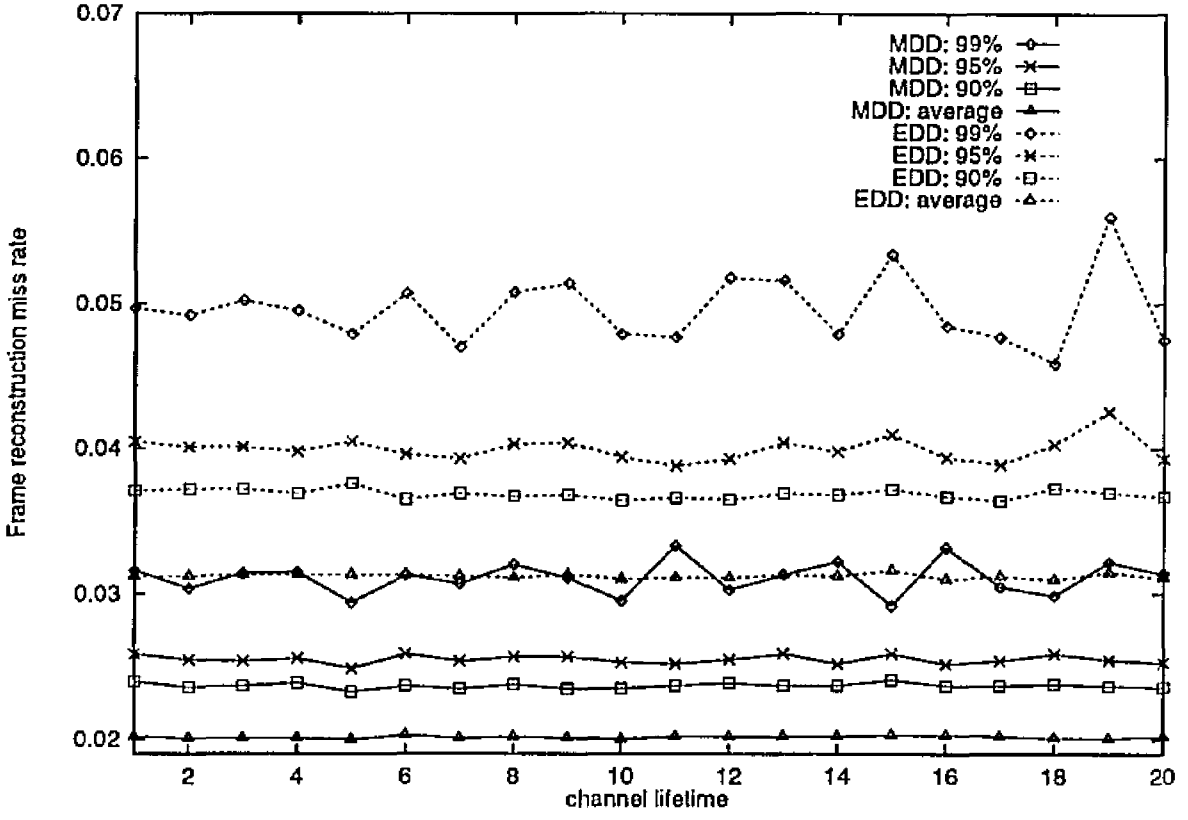



**Figure 3.8:** Frame-reconstruction miss rate for multiplexing 95% channels (short lifetime)

Although the network capacity can only support thirty-four 99% channels, thirty-seven 95% channels, or forty 90% channels by considering the link capacity that needs to be reserved, we still simulate the cases with more channels than the network can support. In these cases, the entire network capacity is reserved and fairly distributed to each channel. By doing this, in Fig. 3.10, we may find that the 35th and the 36th 99% channels, and the 38th 95% channel may be added and the system can still provide the required performance guarantees. However, these cases are in the region where the network is saturated and the frame-reconstruction miss rate rises sharply due to the insufficient capacity, i.e., the system might not always be able to provide the performance guarantees in these cases.

In addition to the frame reconstruction rate, the distribution of miss frames is also an important measure of picture quality. For example, under the same frame-reconstruction rate, consecutive-frame losses usually lead to worse picture quality than uniformly distributed frame losses. Fig. 3.11 shows the distribution of frames which can not be reconstructed by their deadlines. The horizontal axis denotes the length of consecutive miss frames, and the

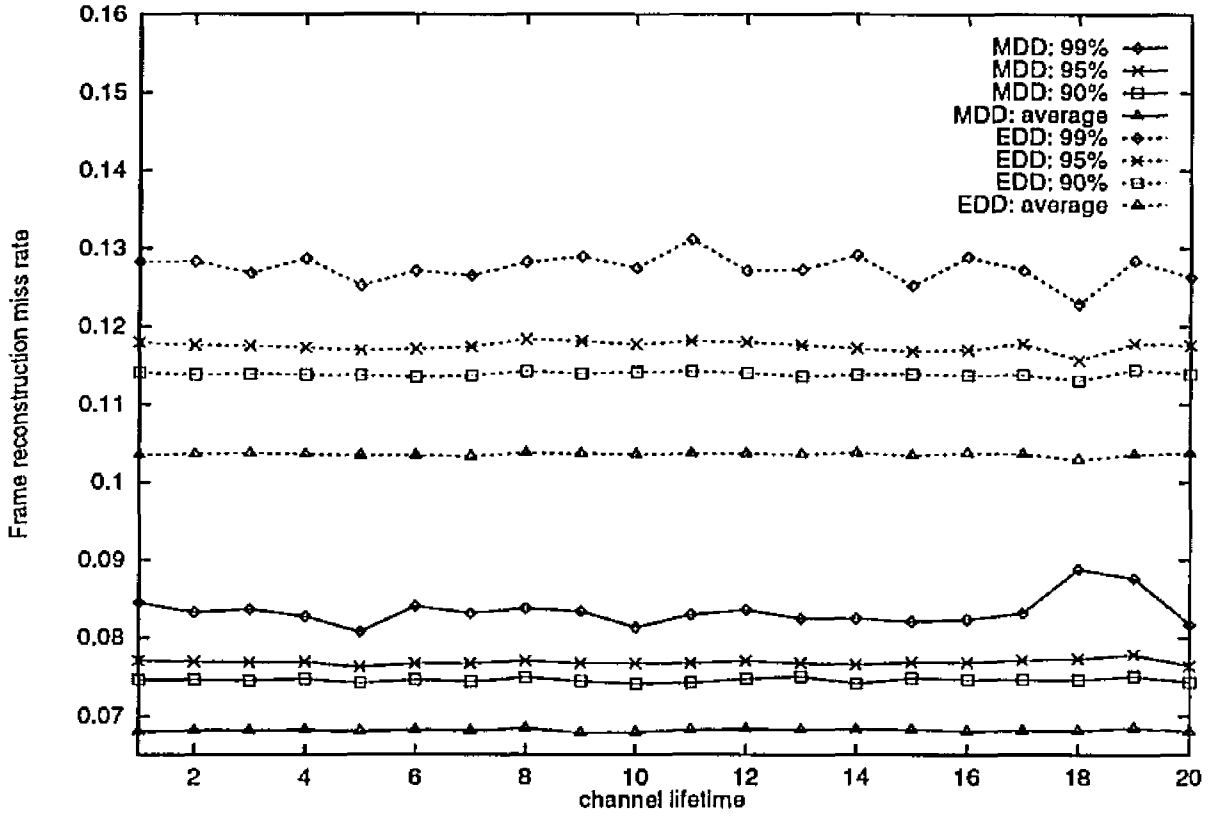



**Figure 3.9:** Frame-reconstruction miss rate for multiplexing 90% channels (short lifetime)

"10" on the horizontal axis represents all miss streaks of length 10 or more. The vertical axis shows the percentage of frames falling in a certain length of miss streak among all *missed* frames. As can be seen from the figure, most (more than 90%) of lost frames under MDD are "1-misses" (miss streak of length one), and there are virtually no "6-misses". At the picture rate of 30 frames per second, these 1-misses are usually invisible by human eyes. By contrast, less than 50% of missed frames are 1-misses under the normal EDD. Due to the loss of I-frames, more than 50% are 8-misses. Since losing 8 (or more) consecutive frames implies more than 0.25 seconds of picture loss, these 8-misses will lead to visible jitters. From the distribution of missed frames, the performance of MDD is much better than that of EDD.

The simulation results show that the proposed channel-multiplexing strategy can significantly improve the network utilization (shown in Fig. 3.6). In addition, it also shows that the normal EDD is not able to handle video streams with frame inter-dependency and the proposed MDD scheduling algorithm with the node-based scheme can provide the promised

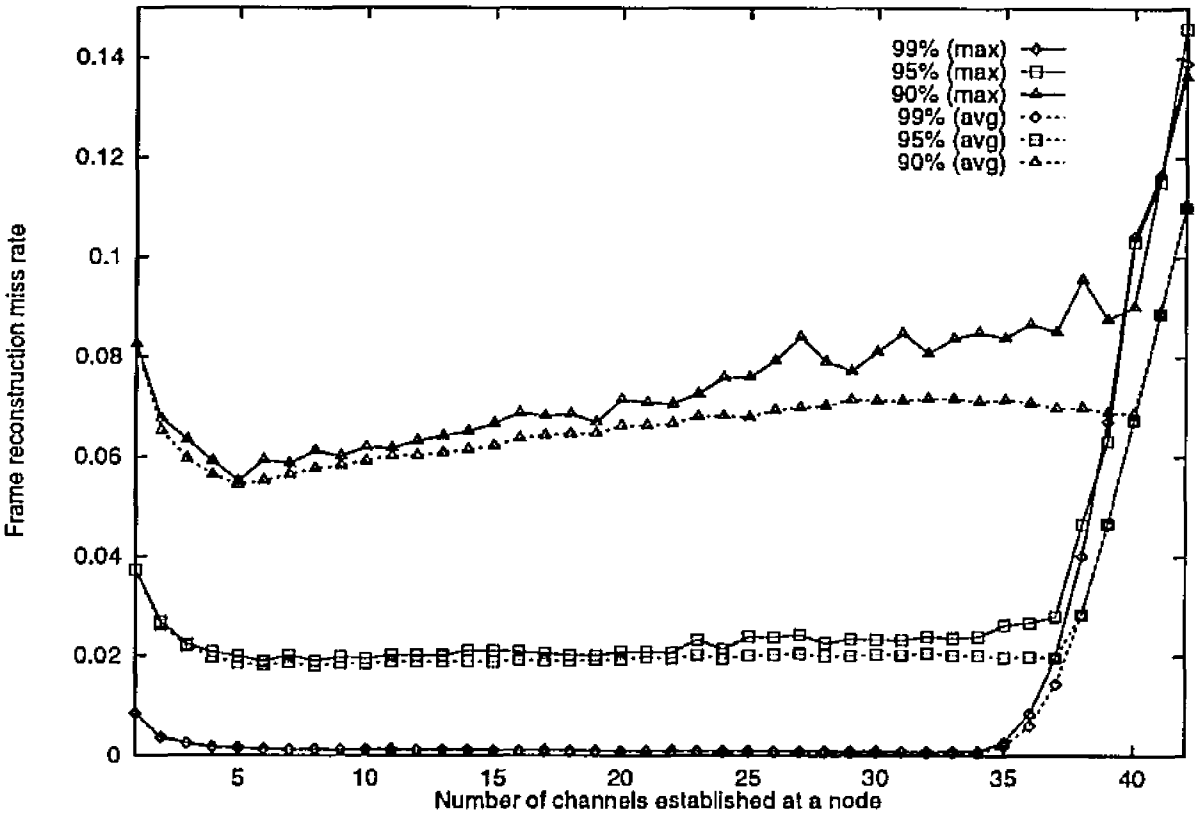



**Figure 3.10:** Frame-reconstruction miss rate for long lifetime channels

performance guarantees and better distribution of missed frames.

## 3.5 Conclusion

We presented a node-based channel-multiplexing strategy and a scheduling algorithm (MDD) which can provide performance guarantees for real-time channels with non-independent frames. With the given traffic-generation characteristics and performance requirements, the node-based channel-multiplexing strategy can (i) reduce the link capacity that needs to be reserved and (ii) preserve the ability of independent addition and deletion of real-time channels, which is of practical importance.

By integrating with the MDD algorithm, the node-based scheme is applicable to applications which generate real-time streams with non-independent frames, such as compressed video applications. Simulation results show that the combination of MDD and the node-based scheme is very effective in reducing the link capacity that needs to be reserved to the level of average real-time traffic from the original worst-case level of traffic even in the presence of frame-dependency. This reduction is practically important since the capacity

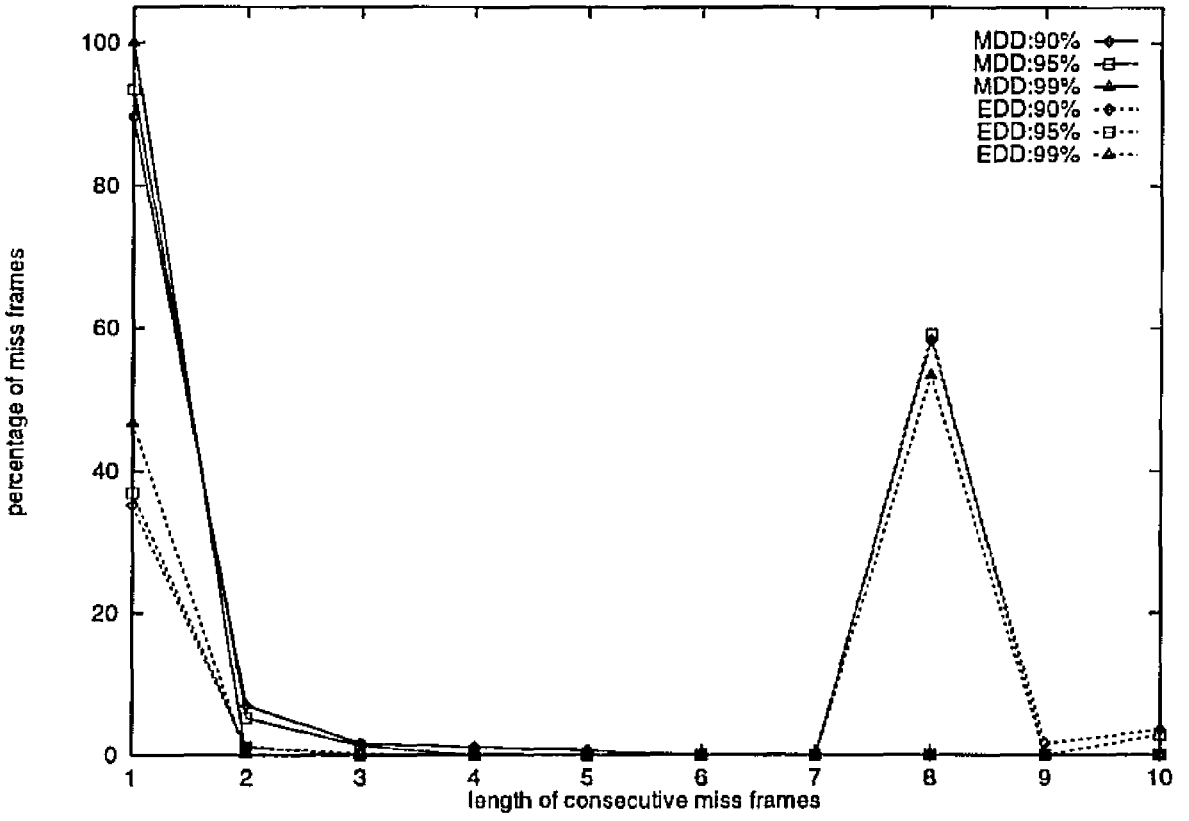



**Figure 3.11:** Distribution of miss frames

reserved in the worst case is often significantly larger than that in an average case.

# CHAPTER 4

# A DISTRIBUTED ROUTE-SELECTION SCHEME FOR ESTABLISHING REAL-TIME CHANNELS

## 4.1 Introduction

In Chapters 2 and 3, we proposed schemes which can provide real-time communication services with performance guarantees on a multiaccess network. However, in order to support real-time communication between nodes which are not connected directly via a multiaccess network, in this chapter, we will address the route-selection problem for multihop real-time channels which can be used to expand the multiaccess network solutions to wide-area point-to-point networks and/or multiple interconnected multiaccess networks.

The concept of real-time channels proposed by Ferrari and Verma [18] is used to provide real-time communication with performance guarantees for point-to-point networks, although we adopt the same terminology for our schemes in the multiaccess network environment. Generally, two distinct phases are required to realize the concept of real-time channel: off-line channel establishment and run-time message scheduling. The channel-establishment phase is of prime importance to the realization of a real-time channel, and during this phase, the system must select a route between the source and destination of the channel along which sufficient resources can be reserved to meet the user-specified delay and buffer requirements. Although several channel-establishment schemes have been proposed in the literature [12–14, 18, 22, 23, 38], very few of them have explicitly addressed the issue of selecting a route between the source and destination of a channel, despite its importance to the channel-establishment phase.

Since the number of possible routes between two communicating peers could be large, selecting a route for each real-time channel is potentially a time-consuming task. It is therefore very important to develop an efficient scheme that is guaranteed to select a "qualified"

route, if any, for each requested real-time channel. If the worst-case anticipated traffic over a real-time channel is given (typically in terms of the minimum message inter-arrival time and the maximum message size), a "qualified" route for this real-time channel is defined to be a route which can meet the user-specified end-to-end delay requirement without compromising any of the existing guarantees. The service provider (the network operating system in our case) must also be able to reject a channel-establishment request as soon as possible if no qualified routes are available for the requested channel.

There are basically two approaches to the route-selection problem: centralized or distributed. Most existing channel-establishment schemes are based on the centralized approach [12, 13, 22, 23, 38]. They simply assume the existence of a global network manager which maintains information about all established real-time channels, the topology and resource distribution and commitment of the network, and can thus select an appropriate route for each real-time channel requested. In such a centralized scheme, all real-time channel-establishment requests require the network manager's approval. That is, each channel-establishment request is sent, along with its traffic-generation characteristics and user-specified performance requirements, to the network manager, which then selects a qualified route and reserves resources along the selected route. The network manager also informs all intermediate nodes on this route of the establishment of the new channel and the information necessary for run-time scheduling of the messages of this channel. Although with the centralized approach one can devise efficient algorithms for the network manager to select qualified routes, there are two serious problems with this approach. First, the network manager is likely to be a performance bottleneck, since it must handle all channel establishment and disconnection requests. Second, the system is susceptible to single-point failures of the network manager, since without the network manager no new real-time channel can be established.

In contrast with the centralized approach, the distributed route-selection approach can avoid performance and reliability bottlenecks. However, it generally suffers the following inefficiency problem. Since there could be many possible routes between two communicating peers, it may be too time-consuming to search every possible route and perform an admission test on each route during the channel-establishment phase. On the other hand, if we only test a small number of routes, we may not find a qualified route even if there exists one. That is, the distributed approach may reject a channel-establishment request even when it would have been accepted if more routes had been checked.

To choose a qualified route for a new real-time channel, we have to perform an admission test on each route to check if there are sufficient resources along the route to meet the user-specified end-to-end delay requirement for this channel. (Although there may be other performance requirements to be met, for clarity of presentation we will focus on meeting the user-specified end-to-end delay requirement.)

Since there could be a large number of routes between two communicating peers, choosing a qualified route among all possible routes between the source and destination of each requested channel may not be an easy task. There are two simple-minded approaches to the distributed route-selection problem:

1. Sequential search of all possible routes one by one, or $K$ routes at a time.

2. Parallel search of all possible routes, i.e., sending multiple copies of an establishment-request message through all possible routes, making "conditional" resource reservation and performing admission tests on all of them.

The second approach is practically infeasible due to its excessive operational overhead. The first approach, on the other hand, could be potentially time-consuming for the complete search of all possible routes, and its operational overhead is proportional to $K$.

To guarantee the discovery of a qualified route, if any, we have to search all possible routes between the source and destination of a channel to be established, while keeping the operational overhead low enough to make the scheme practically feasible. In the next section, we will propose a scheme that satisfies the above requirement for a single establishment request at a time. The scheme also works well for multiple simultaneous requests if the existing real-time traffic load is reasonably low. Basically, each node in the network maintains certain information of the real-time traffic going through it and exchanges the information with its neighbors, so that the Bellman-Ford shortest path algorithm can be used to guarantee a qualified route can be found, if any. Although the proposed scheme starts with searching all possible routes at the same time, it prunes infeasible routes quickly. Under the assumption that messages travel faster through lightly-loaded link, its worst-case operational overhead is only a linear function of $E$, the number of links in the network.

The chapter is organized as follows. Our proposed solution to this problem and its overhead analysis are presented in Section 4.2. In Section 4.3, we demonstrate via examples the effectiveness of the proposed solution. The chapter concludes with Section 4.4.

## 4.2 The Proposed Solution Approach

We first describe the environment and the assumption under which our distributed route-selection scheme will be developed. The underlying network is an arbitrary point-to-point network. As in [7,15,23,24,44], the generation of real-time messages is assumed to be governed by the linear-bounded model that is characterized by three parameters: maximum message size $S_{max}$ (bytes), maximum message rate $R_{max}$ (messages/second), and maximum burst size $B_{max}$ (messages). In the linear bounded model, there are two restrictions on each arrival:

- The number of messages generated in any time interval of length $t$ does not exceed $B_{max} + tR_{max}$.
- The length of each message does not exceed $S_{max}$.

Based on this message arrival model, the authors of [23,24] proposed a scheme to compute the worst-case delay on each link and a run-time scheduling algorithm for real-time messages. By adding the worst-case delays of all links that a channel runs through, one can calculate the worst-case end-to-end delivery delay. This end-to-end delay is then compared against the user-specified end-to-end delay bound for the requested channel and the system can decide whether to accept/reject the corresponding real-time channel-establishment request. Note that these schemes have been developed under the assumption that a proper route for the requested channel was already available. Using the delay-estimation method in [23,24] and a Bellman-Ford-like algorithm, we will in this chapter develop a scheme to find a qualified route for each channel-establishment request.

### 4.2.1 Link-Delay Estimation

Since real-time messages are given priority over non-real-time ones, we will ignore the effects of non-real-time traffic in the rest of this chapter unless stated otherwise. We will thus assess the delay of a link based only on the underlying real-time traffic. Since the algorithm in [23,24] will be used to compute link delays, we will briefly introduce this algorithm first.

The goal of the algorithm in [23,24] is to compute the minimum worst-case delay on a link for a new real-time channel to be added without compromising the performance guarantee of any of the existing channels on the link. Let $\{M_i = (C_i, p_i, d_i), i = 1, \ldots, k\}$ be the set of $k$ existing channels on a link, where $C_i$ is the maximum time required to

transmit a message of channel $M_i$ on the link, $p_i = I^i_{min} = \frac{1}{R^i_{max}}$ is the minimum message inter-arrival time in $M_i$, and $d_i$ is the maximum delay assigned to $M_i$ on this link, or *link (delay) deadline*. Note that the inequality $d_i \leq p_i$ must hold for the algorithm in [23,24] to work correctly. Given a new channel $M_{k+1} = (C_{k+1}, p_{k+1})$ to be established, the authors of [23,24] proposed an algorithm for computing the minimum worst-case response time (MWRT), $\tau_{k+1}$, on a link of channel $M_{k+1}$'s route without compromising the performance guarantees of other existing channels. The algorithm statically assigns priority to each real-time channel to calculate the MWRT for this new channel, but uses an Earliest-Due-Date (EDD) algorithm for run-time scheduling. The algorithm can compute the MWRT for a new channel through link $\ell$ based on the traffic-generation characteristics ($C$ and $p$) of the channel, when $C$ (maximum service time for a message), $p$ (minimum message inter-arrival time) and $d$ (maximum permissible delay over link $\ell$) are available for all existing channels.

The method in [23,24] has *not* included those channels pending for final confirmation in the calculation of MWRT for the new channel-establishment request, but we will include them in our calculation of MWRT as if they had already been established. This can simplify the channel-establishment phase, since the MWRT remains valid when the confirmation message travels back from the destination to the source. Otherwise, the MWRT for a new channel may change due to the confirmation of other pending channels which share one or more links with this channel, and thus, we have to check this possibility at every intermediate node the confirmation message visits en route to the source node. On the other hand, inclusion of these pending channels in the link-delay estimation will sometimes make MWRT larger than what it actually would be if some of them are rejected or choose not to use this link later. This over-estimation of MWRT may result in incorrect rejections of channel-establishment requests. Fortunately, the over-estimation problem occurs only when two requests are initiated at about the same time. The incorrect rejection decisions due to the over-estimation of MWRT will be made only when there is a very high percentage of real-time traffic so that the over-estimation of MWRT may make the end-to-end delay larger than the latency required by the application. Since a good system design should also anticipate the existence of a substantial percentage of non-real-time traffic, the over-estimation problem is usually not serious. In order to avoid any possible confusion, "existing channels" will henceforth mean *both* established and pending channels in Chapter 4 and 5.

Note that different real-time channels have different traffic-generation patterns, and hence, each of them is associated with a different MWRT, i.e., different channels may have

different MWRTs over the same link. Determination of each channel's MWRT on a link will be referred to as *link-delay estimation*. In Section 4.3, we will present an example (Example 4.2) which includes the illustration of the link-delay estimation procedure.

### 4.2.2 The Route-Selection Algorithm

Based on the above definition of link delay, we can apply the Bellman-Ford algorithm [9,46] to solve the route-selection problem. Note that the proposed algorithm is not exactly like the original Bellman-Ford shortest path algorithm in terms of the number of routes that are explored. Under the original Bellman-Ford algorithm, only the one which has the shortest delay is explored at any time. However, under our algorithm, we explore *all* routes which are possible to be the shortest path at the same time.

Since the information of existing channels is necessary for the calculation of a new channel's MWRT as well as for the run-time scheduling of messages belonging to those channels already established, each node has to maintain two sets of tables for existing channels. The first set is the tables of established channels (TECs), one for each of its outgoing links. Each entry of a TEC represents a real-time channel which goes through the corresponding link and consists of the following four data fields.

- Channel identifier (ID) which uniquely identifies the corresponding real-time channel. In order for a source node to generate unique channel IDs, each ID consists of two parts. The first part is the source ID (or address), and the second part is a channel number (unique within the source). This composition of channel IDs ensures their uniqueness throughout the network.
- The maximum service time of a message ($C$) of this channel.
- The minimum message inter-arrival time ($p$) of this channel.
- The maximum permissible delay on this link ($d$) for this channel.

To be consistent with the way channel priorities are assigned for the link-delay estimation [23,24], these entries are placed in ascending order of $d$ values, i.e., the highest priority is given to the channel with the least permissible delay on this link. Note that this priority assignment is used only for calculating MWRT; a multi-class EDD algorithm is used for the run-time scheduling of message transmissions. (The optimality of EDD in meeting deadlines legitimates the off-line calculation of MWRT with fixed-priority scheduling followed by the

on-line EDD scheduling of message transmissions.)

The second set of tables each node has to maintain are "temporary" tables for pending channel-establishment requests, also one for each of its outgoing links. These tables will be referred to as "tables of pending requests" (TPRs). Each entry of a TPR represents a channel-establishment request (or a pending channel) and consists of six fields. The first three fields are the same as those of a TEC and the remaining three fields are:

- $d^a$: the accumulated delay from the source to the current node,
- *timeout*: the expiration time of this request message,
- $r$: MWRT of the corresponding outgoing link.

For a real-time channel-establishment request, the first five fields are the same for all outgoing links of a node, and thus, can be shared among all TPRs for the node's different outgoing links, i.e., one may easily use only one table to store the combined information of all TPRs. However, for convenience of presentation, we will assume that each TPR (one per outgoing link) contains all of these data fields.

When the source wishes to establish a real-time channel to another node, it will use the link-delay estimation method described earlier to compute the channel's MWRT on each of its outgoing links. After computing all MWRTs, the source will send a real-time channel request message (*Req*) via each outgoing link, which contains a channel identifier (*ID*), the destination address (*destination*), the maximum message size of this channel ($S_{max}$), the minimum message inter-arrival time ($p$), the end-to-end delay bound $D$, the expiration time (*timeout*) of this request, the path (*path*) and the total number of hops (*hops*) this message has traveled thus far, and the corresponding accumulated delay $d^a$. Initially, the $d^a$ field is set to the MWRT of the corresponding link, *path* is set to the source and *hops* is set to 1. Note that although we include *hops* in the request message for convenience of presentation, it can be omitted in a real implementation because the information carried in *hops* can be derived from *path*. Copies of this channel-establishment message will be put into the queues of all of its outgoing links at the same time,[1] each with priority lower than all existing channels but higher than non-real-time traffic. This new establishment request will also be inserted into the source node's TPRs.

[1] One can build hardware to do this [17]. If such hardware is not available then the copies will be put in the queues sequentially, one at a time.

```
Procedure rcv_req
If (Req.timeout ≤ current_time) then discard_req;
else if (Req.ID ∈ TEC) then discard_req;
else if (Req.destination = A) then reply_req;
else if (Req.ID ∈ TPR) then {
        if (Req.d^a ≥ TPR(Req.ID).d^a) then discard_req;
        else {
          TPR(Req.ID).d^a := Req.d^a;
          forward_req;
        }
}
else {   ;;;(Req.ID not in TPR)
        r := compute_MWRT;
        if (Req.d^a + r < Req.D) then { insert_req(r); forward_req;}
        else discard_req;
}
```

**Figure 4.1:** Procedure of processing a channel-establishment request

Fig. 4.1 outlines the procedure an intermediate node A will execute when a real-time channel-establishment request is received. Procedure *rcv_req* checks the received request message to determine whether the message should be discarded or processed further. The first two **if** statements check whether the request has expired ($timeout \leq current\_time$) or the request is in any of TECs, i.e., a qualified route for the channel has already been found. If either of these is true, the request message will be discarded. Procedure *reply_req* will then be called if node A is the destination of the channel.

The fourth **if** clause is for the requests already in TPRs. The request will be discarded if the accumulated delay ($d^a$) of the received message is not smaller than the corresponding one in TPRs which represents the minimum accumulated delay from the source known to node A thus far. If the $d^a$ value of the received message is smaller, node A will update its TPRs to reflect the fact that a better route has been found and the received request will be forwarded to the next node by procedure *forward_req*.

Finally, we conclude that the request is new to node A. Thus, the request message will be appropriately stored and forwarded (with procedure *insert_req* and *forward_req*)

**Procedure** *insert_req*($r$)

$TPR.ID := Req.ID$;
$TPR(Req.ID).C := Req.S_{max}/link\_speed$;
$TPR(Req.ID).p := Req.p$;
$TPR(Req.ID).d^a := Req.d^a$;
$TPR(Req.ID).timeout = Req.timeout$;
$TPR(Req.ID).r := r$;

**Procedure** *forward_req*

$Req.d^a := TPR(Req.ID).d^a + TPR(Req.ID).r$;
$Req.hops := Req.hops + 1$;
;;; concatenate A and *Req.path*.
$Req.path := A \cdot Req.path$;
;;; all other fields remain the same.
forward this request message to all neighbors except B.

**Figure 4.2:** Procedures of inserting and forwarding a request

if the sum of the accumulated delay and the MWRT of the corresponding next link is less than $D$. Otherwise, the path this request message has traveled so far cannot possibly be a qualified one, so it will be discarded. Consequently, a request message will be forwarded by an intermediate node only if it carries a smaller accumulated worst-case response time ($Req.d^a$) before its expiration time.

Fig. 4.2 shows the procedures for inserting a new channel-establishment request and forwarding a request. As can be seen from these procedures, most of the fields are directly copied from the requesting messages to TPR and the forwarding messages. Note that the request message is assumed to come from the immediate upstream node $B$.

Each destination node has to keep a temporary list of already-processed requests (LPRs) in order to avoid reporting the request to applications more than once. Each entry of this list consists of two fields, request $ID$ and *timeout* which tells when to discard the request. Fig. 4.3 shows the operations a destination node will perform after receiving a channel-establishment request. From Procedure *reply_req*, one can see that if the system decides to accept the channel-establishment request, the (qualified) path carried by the request that arrived first will be selected as the route for the real-time channel.

Since the $d^a$ field of a channel-establishment request represents the sum of MWRTs of all links on the path from the source to destination, the user-specified end-to-end delay bound

**Procedure** *reply_req*

**If** ($Req.ID \in LPR$) **then** *discard_req*;

**else** { ;;;(insert a new entry to LPR)
$LPR.ID := Req.ID$;
$LPR(Req.ID).timeout := Req.timeout$;
**If** (*the application accepts the request*) **then** *send_reply*(*accept*);
**else** *send_reply*(*reject*);

}

**Figure 4.3:** Procedure of processing a channel-establishment request at the destination

$D$ may be larger than $d^a$, i.e., we are allowed to spend more time than the corresponding MWRTs when sending a message across each intermediate link. In such a case $D - d^a$ will be divided evenly into *hops* parts at the destination and distributed to all links along the path [23,24]. The permissible delay of a real-time message of this particular channel over an intermediate link — simply called the *link (delay) deadline* — is the channel's MWRT of that link plus $(D - d^a)/hops$. Since this sum is stored in the table of existing channels ($d$ field in TEC) and used for run-time scheduling, $(D - d^a)/hops$ is included in the channel-establishment confirmation message (by procedure *send_reply*(*accept*)) from the destination to the source via the same path the corresponding request message had traveled (but in the opposite direction). Let *Reply* denote a channel-establishment confirmation message which consists of four fields: $ID$, $flag$ (accept or reject), $diff$ ($= (D - d^a)/hops$) and $path$ (the remaining path back to the source node). Fig. 4.4 shows how a positive confirmation message is constructed (*send_reply*(*accept*)), and the operations the intermediate nodes will perform when receiving a (positive or negative) reply message ($forward\_reply$). Note that *head*(*list*) represents the first element of *list*, and *tail*(*list*) represents the remaining list after *head*(*list*) is removed from *list*.

The operations necessary to keep these route-selection tables (mainly TECs) up-to-date during the channel-disconnect phase are very simple. We require one of the two communicating peers to send a disconnect message through the route of the real-time channel to the other communicating peer. In this disconnect message, only the channel $ID$ needs to be included. All the intermediate nodes will delete the corresponding entries in their TECs upon receiving the disconnection message. Thus, we do not consider the load of this real-time channel in all subsequent MWRT estimations.

**Procedure** *send_reply(accept)*

*Reply.ID* := *Req.ID*;
*Reply.diff* := (*Req.D* − *Req.d*$^a$)/*Req.hops*;
*Reply.flag* := 1;
*Reply.path* := *tail*(*Req.path*);

**Procedure** *forward_reply*

**If** (*Reply.flag* = 0) **then** *TPR*(*Reply.ID*).*d*$^a$ = 0;

**else** {

;;; move the channel from TPR to TEC
copy the ID, C and p fields from TPR to TEC.
*TEC*(*Reply.ID*).*d* := *Reply.diff* + *TPR*(*Reply.ID*).*r*;
delete the corresponding entries in all TPRs of node A.

}

*next* := *head*(*Reply.path*);

*Reply.path* := *tail*(*Reply.path*);

forward this reply message to the next upstream node *next*.

**Figure 4.4:** Procedure of handling reply messages

### 4.2.3 Performance and Overhead Analysis

The first goodness measure we are interested in is the "completeness" of the proposed scheme, i.e., whether the scheme is capable of finding a qualified route, if any. For a single request, the Bellman-Ford (shortest-delay path) [9,46] algorithm can ensure the least MWRT path to be found, although other larger-delay routes may be found first when the request messages happen to travel faster via these routes than via the least MWRT path. As mentioned before, our algorithm is not exactly like the original Bellman-Ford shortest path algorithm in terms of the number of routes that are explored. Under the original Bellman-Ford algorithm, only the one which has the shortest delay is explored at any time. However, under our algorithm, we explore *all* routes which are possible to be the shortest path at the same time. Thus, if the least MWRT path is not "qualified" for the requested channel, then there is no qualified route available for the channel. Since the least MWRT path can always be found with the Bellman-Ford algorithm, the proposed scheme is complete for the single-request case. However, due to the over-estimation of MWRTs when there are multiple simultaneous requests, the proposed scheme may not be complete, especially

when the network has a high percentage of real-time traffic so that the over-estimation of MWRTs will make the end-to-end MWRT delay larger than the user-specified end-to-end delay bound. (As we discussed in the delay-estimation procedure, the over-estimation problem is usually not serious in practice. In the next section, we will provide an example to illustrate an over-estimation situation.)

Another performance measure is the time needed to establish a channel. For a single request, the worst-case time needed to accept an establishment request is the time for the request message to travel from the source to destination then back to the source node via the least MWRT path. This is a round-trip delay between the source and destination via the least MWRT path. In general, if a qualified path can be found (may not necessarily be the least MWRT path), the time to complete the corresponding channel-establishment request is the time for the request message to travel from the source to destination then back to the source via the qualified route found first.

The primary overhead incurred in the proposed channel-establishment procedure is the number of times (copies) a request message has to be transmitted for each channel establishment request. Note that "one time (copy)" is defined as "sending a message across one link", e.g., a message is said to be transmitted $n$ times if the message is sent across $n$ hops. For more accurate estimation, the request message is assumed to travel faster through a lightly-loaded link (while considering only real-time traffic). This assumption generally holds as the priority of the request message is lower than real-time traffic but higher than non-real-time traffic. Under this assumption, we may find that each node will send a request message to its neighbors only once, since the request message will be forwarded only when the conditional statement ($Req.d^a \geq TPR(Req.ID).d^a$) in Fig. 4.1 is false. A node will likely receive the copy of a request message which travels through the route with the smallest value of $Req.d^a$ first, and thus, all subsequent copies of the same request message will be discarded. Under this assumption, a request message will therefore be transmitted at most $2K$ times in the worst case, i.e., each node sends a copy of the request message to all its neighbors once, where $K$ is the number of links in the network.

The conditional statement ($Req.d^a + r < Req.D$) in Fig. 4.1 is used to stop the unnecessary propagation of a request to a region where no qualified routes exist. Since a request will be inserted in TPR and forwarded through an outgoing link only when the sum of $Req.d^a$ and the MWRT over the outgoing link is smaller than $D$ (=$Req.D$), the request message will not propagate too far, i.e., the nodes whose distance from the source (in terms

of the worst-case delivery delay for this new channel) is greater than $D$ will not receive the request message. Although this fact may not improve the *worst-case* overhead, for most of the time it makes a significant reduction of overhead.

We may also put a restriction on *hops* to reduce the overhead; we may stop forwarding a request if the number of hops the message has traveled so far is more than a pre-defined limit. Nodes which are too far (in terms of hop count) from the source $N$ will not receive the requests sent by $N$. If the pre-defined limit can be chosen appropriately, this method may significantly improve both the worst-case overhead and the actual performance. However, since this method also reduces the chance of finding a qualified route, it is not included in the proposed solution and only mentioned as a possible way to reduce the overhead.

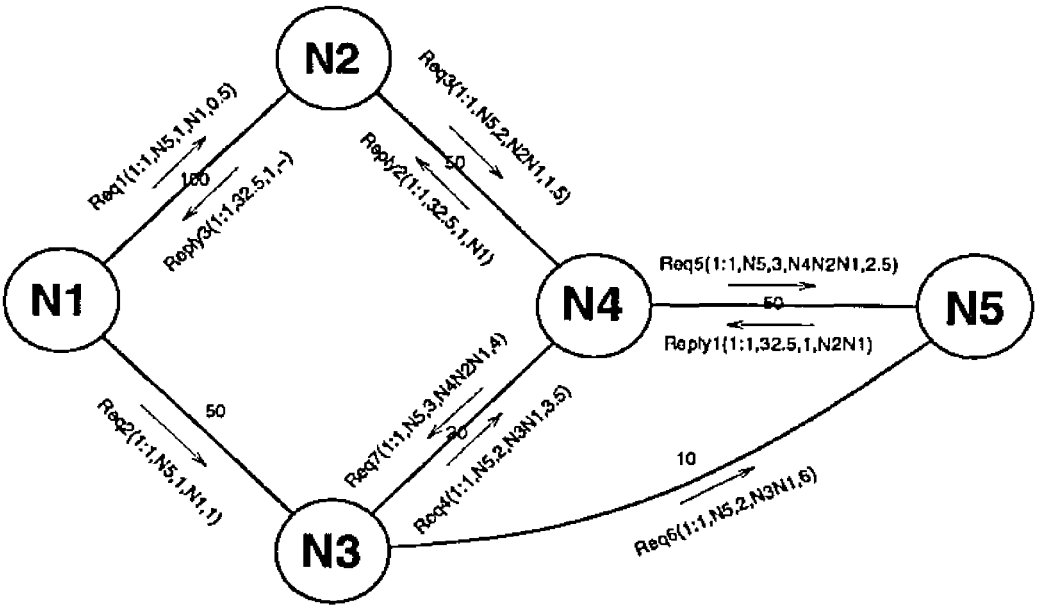
## 4.3 Examples

In this section, we will present several examples to illustrate the operations and utility of the proposed route-selection scheme under various conditions.

**Example 4.1:** Fig. 4.5 shows a small network with five nodes. Each link is labeled with a number representing its transmission bandwidth in million bits per second (Mbps); for example, the link between N1 and N2 is a 100 Mbps full-duplex link. We will illustrate all the operations in the network for establishing the first channel with $S_{max} = 50 Kbytes$ (maximum message size), $p = 50\,ms$ (minimum inter-arrival time), and $D = 100\,ms$ (end-to-end delay bound). The source of this channel is N1 and its destination is N5. We thus assign 1:1 as the channel's $ID$. The first "1" represents the source node, and the second "1" represents a number which is unique to the source node N1. Since no real-time channel exists at this time, all TECs, TPRs, and LPRs are empty.

Upon reception of a channel-establishment request, the maximum service time ($C$) for the messages of this channel can be computed by dividing the maximum message size by the link bandwidth. $C = 0.5\,ms$ for link $1 \rightarrow 2$ and $C = 1\,ms$ for link $1 \rightarrow 3$. Since there is no other real-time channel, the MWRT on a link equals its $C$ value. N1 stores this request in its TPR:$1 \rightarrow 2$ and TPR:$1 \rightarrow 3$. In Fig. 4.5, we omit $C$ and $p$ fields because $C = r$ and $p$ are the same in all TPRs. *Timeout* is also omitted for clarity of presentation.

N1 sends two request messages (Req1 and Req2) to N2 and N3, respectively. Although there are 9 fields in a request message, only $ID$, *destination*, *hops*, *path* and $d^a(ms)$ (in this




TPR 1->2
ID da r
1.1 0 0.5

TPR 1->3
ID da r
1:1 0 1

TPR 2->4
ID da r
1:1 0.5 1

TPR 3->4
ID da r
1:1 1 2.5

TPR 3->5
ID da r
1:1 1 5

TPR 4->5
ID da r
1:1 1.5 1

TPR 4->3
ID da r
1:1 1.5 2.5

TEC 4->5
ID C p d
1:1 1 50 33.5

TEC 2->4
ID C p d
1:1 1 50 33.5

TEC 1->2
ID C p d
1:1 1 50 33

LPR 5
ID
1:1

**Figure 4.5:** Example 4.1: a simple five-node network.

order) are shown in Fig. 4.5, since other fields are the same for all request messages. For example, Req1(1:1,N5,1,N1,0.5) represents $ID$ =1:1, *destination* =N5, *hops* = 1, *path* =N1 and $d^a = 0.5\,ms$.

After N2 and N3 receive Req1 and Req2, respectively, these messages will be stored and the $C$ field for link 2 → 4, link 3 → 4 and link 3 → 5 will be computed. Since there is no other channel, each $C$ is equal to the MWRT on the corresponding link. Thus, the entries for this message in TPR:2 → 4, TPR:3 → 4 and TPR:3 → 5 can be inserted and the corresponding request messages can also be sent. As can be seen in Fig. 4.5, *path* "grows" to N2N1, *hops* increments by 1 and $d^a$ becomes 1.5 in Req3, which is sent to N4 by N2. Req4 (N3 → N4) and Req6 (N3 → N5) can also be generated in the same manner.

Due to the randomness of network traffic, it is not certain whether Req3 or Req4 will arrive at N4 first. However, since the path N1N2N4 has a smaller MWRT and channel-establishment requests are given priority over non-real-time traffic, Req3 will very likely arrive at N4 first. So, in Fig. 4.5, we assume Req3 arrives at N4 before Req4. N4 will thus

process the request based on Req3 and the entries in TPR:4 → 5 and TPR:4 → 3 can be obtained (as shown in Fig. 4.5) and the corresponding request messages (Req5 and Req7) can be sent.

When Req4 arrives at N4, the $d^a$ carried in Req4 will be compared with the $d^a$ value stored in N4's TPRs. Since the $d^a$ (=3.5) carried by Req4 is greater than that (=1.5) of N4's TPRs, Req4 will be discarded.

At N5, a similar situation will occur. We assume that Req5 will arrive at N5 before Req6, because Req5 travels through a path of a smaller MWRT. (We would like to stress, however, that Req6 may possibly arrive at N5 before Req5.) Since N5 is the destination of the requested channel and the $ID$ carried by Req5 is not in N5's LPR, the channel's $ID$ will be inserted into N5's LPR and a confirmation/reject message is sent back via the path carried in the request message (N4N2N1). If the reply is a confirmation, we need to compute $diff = (100 - 2.5)/3 = 32.5$. In Fig. 4.5, Reply1 shows a confirmation sent to N4 by N5 with $ID$ =1:1, $flag = 1$, $diff = 32.5$ and $path$ =N2N1.

After receiving Reply1, N4 inserts an entry with $ID$ =1:1, $C = 1\ ms$, $p = 50\ ms$, and $d = 33.5$ (=32.5+1) into TEC:4 → 5. Then all the corresponding entries (with $ID$ =1:1) in N4's TPRs are deleted and Reply2 (with $path = tail(path)$) is sent to the next node specified in Reply1's $head(path)$. Operations at N2 and N1 are similar. □

Before proceeding to more complex examples, we first describe an environment in which such examples will be derived. The same network in Example 4.1 will be used for Example 4.2 (Fig. 4.6), but TECs are no longer empty, i.e., there already exist many real-time channels when a new channel-establishment request is made. Table 4.1 shows the content of all TECs at some time instant. The entries of these tables had been inserted for establishing the following 21 channels. Each channel is described as a 5-tuple $(ID, S_{max}, p, D, path)$. Table 4.2 presents the 21 channels in the order of their establishment. Note that the TECs in Table 4.1 are only one of many *possible* situations after these 21 channel-establishment requests have been processed and accepted. As discussed earlier, due to the randomness of network traffic, there may be many other possible sets of TECs. We will also illustrate such situations in the following examples.

**Example 4.2:** We want to establish a real-time channel with $ID$ =1:8, $S_{max} = 200Kbytes$ and $p = 20\ ms$ in the network of Fig. 4.6 under the environment as specified in Table 4.1. The source of the requested channel is N1 and its destination is N5. We will first let $D = 19$

| ID | C | p | d | ID | C | p | d | ID | C | p | d | ID | C | p | d |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TEC: 1 → 2 | | | | TEC: 2 → 1 | | | | TEC: 1 → 3 | | | | TEC: 3 → 1 | | | |
| 3:4 | 1.0 | 10 | 4.5 | 2:4 | 1.0 | 10 | 4.5 | 2:4 | 2.0 | 10 | 5.5 | 3:4 | 2.0 | 10 | 5.5 |
| 1:6 | 1.0 | 20 | 6 | 5:3 | 1.0 | 20 | 6 | 1:3 | 1.0 | 20 | 8 | 5:2 | 1.0 | 20 | 8 |
| 1:4 | 0.8 | 25 | 9.6 | 4:1 | 0.4 | 25 | 9.8 | 1:2 | 0.6 | 20 | 16.2 | 3:3 | 2.0 | 20 | 20 |
| 1:7 | 2.0 | 10 | 10 | 2:3 | 2.0 | 10 | 10 | 1:5 | 2.0 | 20 | 20 | 4:2 | 1.6 | 50 | 23.8 |
| 1:1 | 0.5 | 50 | 33 | 2:2 | 1.0 | 40 | 19.5 | 2:2 | 2.0 | 40 | 20.5 | | | | |
| TEC: 3 → 4 | | | | TEC: 4 → 3 | | | | TEC: 2 → 4 | | | | TEC: 4 → 2 | | | |
| 1:2 | 1.5 | 20 | 17.1 | 5:4 | 3.0 | 20 | 10.9 | 1:6 | 2.0 | 20 | 7 | 5:3 | 2.0 | 20 | 7 |
| 3:1 | 1.0 | 40 | 20.3 | 5:1 | 3.0 | 20 | 15.9 | 1:4 | 1.6 | 25 | 10.4 | 4:1 | 0.8 | 25 | 10.2 |
| 3:2 | 5.0 | 40 | 21.5 | 4:2 | 4.0 | 50 | 26.2 | 1:1 | 1.0 | 50 | 33.5 | 3:2 | 2.0 | 40 | 18.5 |
| | | | | | | | | 2:1 | 2.0 | 50 | 50 | | | | |
| TEC: 3 → 5 | | | | TEC: 5 → 3 | | | | TEC: 4 → 5 | | | | TEC: 5 → 4 | | | |
| 1:3 | 5.0 | 20 | 12 | 5:2 | 5.0 | 20 | 12 | 1:6 | 2.0 | 20 | 7 | 5:3 | 2.0 | 20 | 7 |
| | | | | | | | | 3:1 | 0.4 | 40 | 19.7 | 5:4 | 1.2 | 20 | 9.1 |
| | | | | | | | | 1:1 | 1.0 | 50 | 33.5 | 5:1 | 1.2 | 20 | 14.1 |
| | | | | | | | | 2:1 | 2.0 | 50 | 50 | | | | |

**Table 4.1:** TECs for Example 4.2

| ID | $S_{max}$ | p | D | path | ID | $S_{max}$ | p | D | path | ID | $S_{max}$ | p | D | path |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1:1 | 50 | 50 | 100 | N1N2N4N5 | 1:2 | 30 | 20 | 33.3 | N1N3N4 | 3:1 | 20 | 40 | 40 | N3N4N5 |
| 2:1 | 100 | 50 | 100 | N2N4N5 | 5:1 | 60 | 20 | 30 | N5N4N3 | 4:1 | 40 | 25 | 20 | N4N2N1 |
| 4:2 | 80 | 50 | 50 | N4N3N1 | 2:2 | 100 | 40 | 40 | N2N1N3 | 1:3 | 50 | 20 | 20 | N1N3N5 |
| 5:2 | 50 | 20 | 20 | N5N3N1 | 3:2 | 100 | 40 | 40 | N3N4N2 | 1:4 | 80 | 25 | 20 | N1N2N4 |
| 3:3 | 100 | 20 | 20 | N3N1 | 1:5 | 100 | 20 | 20 | N1N3 | 1:6 | 100 | 20 | 20 | N1N2N4N5 |
| 5:3 | 100 | 20 | 20 | N5N4N2N1 | 1:7 | 200 | 10 | 10 | N1N2 | 2:3 | 200 | 10 | 10 | N2N1 |
| 2:4 | 100 | 10 | 10 | N2N1N3 | 3:4 | 100 | 10 | 10 | N3N1N2 | 5:4 | 60 | 20 | 20 | N5N4N3 |

**Table 4.2:** The table of previously-accepted requests for Example 4.2.

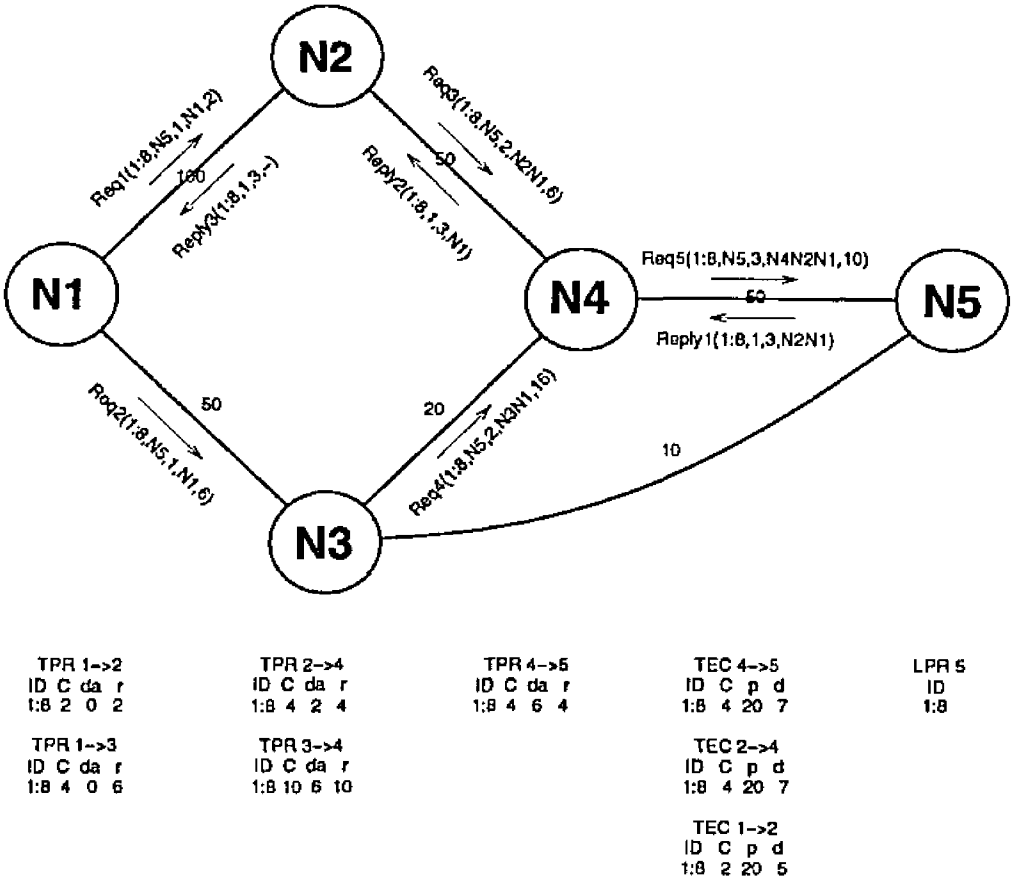



**Figure 4.6:** Example 4.2

to show that only the path N1N2N4N5 is a qualified route for this request. $D$ will then be changed to illustrate other scenarios.

**N1's operations:** The MWRTs ($r$ values in TPR) and $C$ values of N1's outgoing links are computed first. For link $1 \rightarrow 3$, $C = 200/50 = 4$. To compute a channel's MWRT on a link, we need to assign highest possible priority to this channel without violating the link deadlines ($d$ in TEC:$1 \rightarrow 3$) of other existing channels. We start with assigning the requested channel top priority, then check if any of the existing channels' link deadlines will be violated. Recall that the existing channels are placed in a TEC in ascending order of their $d$ values. By giving the newly-requested channel top priority on link $1 \rightarrow 3$, the worst-case delay of channel 2:4 will be

$$4 \times \lceil \frac{5.5}{20} \rceil + 2 = 6 > 5.5.$$

So, we lower the priority of this new request below channel 2:4 but above channel 1:3. With this priority assignment, no existing guarantees will be violated, because the following

schedulability test holds for any channel $i$ whose priority is lower than the requested channel [23, 24, 27].

$$\exists t \in B_i = \{d_i\} \cup \{kp_j : j \in A_i, 0 \leq k \leq \lfloor \frac{d_i}{p_j} \rfloor\},$$
$$W_i(t) = \sum_{j \in A_i} C_j \cdot \lceil \frac{t}{p_j} \rceil + C_i \leq t,$$

where $A_i$ is the set of channels (including the requested channel) whose priority is higher than that of channel $i$, $C_i$ is the maximum service time of channel $i$, $d_i$ is the link deadline of channel $i$, and $p_i$ is the minimum message inter-arrival time of channel $i$. Given below is an acceptable set of $t$ and $W_i(t)$ for channels on link $1 \rightarrow 3$.

- channel 1:3, $t = d_{1:3} = 8$, $W_{1:3}(8) = 7 < 8$.
- channel 1:2, $t = d_{1:2} = 16.2$, $W_{1:2}(16.2) = 9.6 < 16.2$.
- channel 1:5, $t = d_{1:5} = 20$, $W_{1:5}(20) = 11.6 < 20$.
- channel 2:2, $t = p_{1:3} = 20$. $W_{2:2}(20) = 13.6 < 20.5$.

As a result, the priority of the requested channel will be placed between channel 2:4 and channel 1:3. So, the MWRT of the requested channel on link $1 \rightarrow 3$ is equal to the smallest $t$ such that $W_{1:8}(t) = t$, i.e., the smallest $t$ such that $2 \times \lceil \frac{t}{10} \rceil + 4 = t$. Therefore, this channel's MWRT on link $1 \rightarrow 3$ is 6. For link $1 \rightarrow 2$, using the same procedure, the MWRT is computed to be 2 by giving the new channel the highest priority on this link.

Note that this priority assignment for the requested channel is just for computing the MWRT. As discussed in Section 4.2 and in [23, 24], the run-time priority for transmitting the messages of the requested channel, if accepted, is determined based on the channel's link-delay deadline which is greater or equal to the MWRT obtained here and will be determined after the route is selected.

After computing MWRTs, N1 inserts appropriate entries into TPR:$1 \rightarrow 2$ and TPR:$1 \rightarrow 3$ (as shown in Fig. 4.6). Channel-establishment requests are sent to both N2 and N3. Fig. 4.6 shows only five fields in the request messages as in Example 4.1, because the other fields are the same for all request messages.

**Operations of N2 and N3:** After receiving the request message from N1 and determining that this request is new (does not exist in any TPR and TEC), N2 and N3 will perform

similar operations as N1, i.e., compute $C$ and MWRT values for all of their outgoing links except for the links to N1, and store this information in TPRs.

Using the same procedure as above, we get $C = 4$ and MWRT= 4 on link $2 \rightarrow 4$ by assigning the new channel top priority. A request message (Req3 in Fig. 4.6) is sent to N4 with $d^a = 2 + 4 = 6$.

For link $3 \rightarrow 4$, we get $C = 10$ and MWRT= 10 by assigning the new channel top priority. A request message (Req4 in Fig. 4.6) is sent to N4 with $d^a = 6 + 10 = 16$.

For link $3 \rightarrow 5$, we get $C = 20$ and MWRT= 25 by assigning the new channel the lowest priority. However, in order to make the scheme work correctly, we require $d \leq p$ $(= 20)$ on each link of the channel's route [23,24]. So, this link cannot be part of a qualified route, and hence, no request message is sent to N5 via this link and no entry (corresponding to this request) will be inserted into TPR:$3 \rightarrow 5$.

**N4's operations:** Since it is not certain which of Req3 or Req4 will arrive at N4 first, we will discuss both cases.

If Req3 arrives first, then N4 will compute $C$ and MWRT for links to both N3 and N5. For link $4 \rightarrow 3$, we get $C = 10$ and MWRT= 16 by assigning the requested channel the priority lower than channel 5:1 but higher than channel 4:2. So, the accumulated MWRT from N1N2N4N3 is $6 + 16 = 22 > D(= 19)$, and hence, no request message will be sent to N3 and no entry (corresponding to this request) will be added to TPR:$4 \rightarrow 3$.

For link $4 \rightarrow 5$, we get $C = 4$ and MWRT= 4 by assigning top priority to the requested channel. So, N4 stores this information in TPR:$4 \rightarrow 5$ and Req5 (with $d^a = 6 + 4 = 10$) is sent to N5. After N4 receives Req4, since the $d^a$ (=16) carried in the message is greater than that (=10) in N4's TPRs, Req4 is discarded.

On the other hand, if Req4 arrives at N4 first, N4 will compute $C$ and MWRT for links to N2 and N5 based on Req4. For link $4 \rightarrow 5$, because MWRT= 4, the accumulated MWRT, $d^a$, from N1 to N5 will be $16 + 4 = 20$ which is greater than the required end-to-end delay bound 19, so no message will be sent to N5. A similar situation occurs for link $4 \rightarrow 2$, since $C = 4$ and MWRT= 4 (top priority). Thus, when Req3 arrives, the operations performed by N4 will be exactly the same as those in the case when Req3 arrives at N4 first.

**Reply operations:** N5 will perform the same operations as in Example 4.1, but with $diff = (19 - 10)/3 = 3$. The operations to be performed by N4, N2 and N1 when the reply message arrives at these nodes are the same as in Example 4.1.

| | MWRT | | |
|---|---|---|---|
| Path | CH 1:8 first | Same time | CH 2:5 first |
| N1N2N4N5 | 10 | 25.2 | 25.2 |
| N1N3N4N5 | 20 | 27.6 | 36.6 |
| N2N4N3 | NA | 20 | 20 |
| N2N1N3 | 17 | 17 | 8 |

**Table 4.3:** MWRTs for two concurrent requests with $S_{max} = 200 Kbytes$.

If we increase $D$ to 20, the path N1N3N4N5 with the worst-case accumulated delay 20 is a qualified route for the requested channel. Thus, if Req4 arrives at N4 before Req3 (Fig. 4.6), this route will be chosen instead of N1N2N4N5.

On the other hand, if we let $D = 15$ in this example, as can be seen in Fig. 4.6, Req4 will not be sent because its $d^a > 15$. If we decrease $D$ further to 9, then Req5 will not be sent. In this case, no qualified route exists at that time, so the channel-establishment request will time out and, thus, will be rejected. □

**Example 4.3:** In this example, we will use the same network and environment in Example 4.2 to demonstrate the effects of over-estimating link delays in case of multiple pending requests. In addition to the requested channel 1:8 in Example 4.2, we assume another channel ($ID$ =2:5) establishment request occurs at about the same time. This channel is specified as $S_{max} = 200 Kbytes$, $p = 20\,ms$, source = N2 and destination = N3. At first, we ignore the end-to-end delay requirements ($D$) of both channels, i.e., assume both $D$s are sufficiently large.

If these two channel requests arrive sequentially with a sufficiently large inter-arrival time between them (regardless of the order of their arrival), the least MWRT of channel 1:8 is equal to 10 (via path N1N2N4N5), and that of channel 2:5 is equal to 8 (via path N2N1N3). Note that as discussed in Section 4.2, due to the randomness of the network traffic, the path with the least MWRT may not always be chosen. So, the MWRT of a channel depends on which path is chosen. Thus, the least MWRT of a channel is defined to be the MWRT of the path with the smallest MWRT. However, when the inter-arrival time between them is not sufficiently large, i.e., TPRs contain entries for both channels at the same time, the MWRTs (including the least MWRT) may be over-estimated.

| | MWRT | | |
|---|---|---|---|
| Path | CH 1:8 first | Same time | CH 2:5 first |
| N1N2N4N5 | 2.5 | 4.5 | 4.5 |
| N1N3N4N5 | 4.5 | 5.5 | 6.5 |
| N2N4N3 | 7 | 3.5 | 3.5 |
| N2N1N3 | 2.5 | 2.5 | 1.5 |

**Table 4.4:** MWRTs for two concurrent requests with $S_{max} = 50Kbytes$.

Table 4.3 shows the MWRTs of the two requests via two (with the least and the second least MWRT) of their possible routes for different arrival orders of channels 1:8 and 2:5. The first column shows the MWRT when channel 1:8 arrives first and channel 2:5 arrives before the deletion of channel 1:8's entries (in TPRs). The third column shows the opposite, i.e., channel 2:5 arrives first. The second column represents the situation when channel 1:8 request arrives at N1 approximately at the same time as channel 2:5 request arrives at N2. Note that the second column is derived under the assumption that request messages travel faster via links with smaller MWRTs and the first-come-first-serve link scheduling policy for messages of the same priority.

As can be seen from the first and third columns of Table 4.3, the MWRT estimate for the channel which arrives later is unnecessarily large. When two channels arrive at about the same time, the MWRT estimates for both channels are larger than their true value. Note, however, we have not yet considered the end-to-end delay requirement. When $D$ is reasonably large, e.g., $D_{1:8} \geq 26$ and $D_{2:5} \geq 17$, qualified routes for both channels can be found regardless of the over-estimation of link delays. In fact, most real-time applications do not require such short end-to-end delays (at least 100 $ms$ for typical interactive applications) and do not generate such high-volume data streams. For example, if we decrease $S_{max}$ to $50Kbytes$ which is a typical size for multimedia applications, the least MWRT for channel 1:8 is 2.5 $ms$, and for channel 2:5 is 1.5 $ms$. Table 4.4 shows the MWRTs of the same four paths (as in Table 4.3) in this case. As can be seen from Table 4.4, channel 1:8's MWRT increases only by 2 $ms$ in the worst case due to the over-estimation of link delays and channel 2:5's MWRT only by 1 $ms$. Unless the requested channel generates very large messages or it requires a very short end-to-end delay, over-estimation usually does not cause unnecessary denial of channel-establishment requests.

## 4.4 Conclusion

In this chapter, we have proposed an efficient distributed route-selection scheme which is guaranteed to find a qualified route, if any, for each single real-time channel-establishment request. By equipping two simple tables with each node, the proposed scheme can not only eliminate the common reliability and performance bottlenecks of centralized route selection, but also keep the operational overhead sufficiently low for practical use.

The proposed scheme is presented in procedure form, and its correctness and completeness are discussed. In the next chapter, we will introduce a specialized scheme which can solve the route-selection problem by a simple table look-up.

# CHAPTER 5

# A DISTRIBUTED TABLE-DRIVEN ROUTE-SELECTION SCHEME FOR ESTABLISHING REAL-TIME CHANNELS

## 5.1 Introduction

In Chapter 4, we propose a *generic* distributed route-selection scheme which is guaranteed to find a qualified route, if any, for each single real-time channel-establishment request. However, this scheme suffers the problem of over-estimating link delays. That is, when there are multiple *simultaneous* channel-establishment requests, the link delays may be over-estimated as a result of treating the pending channels as if they had already been established, thus perhaps incorrectly rejecting channel-establishment requests; they could have been accepted if link delays had not been over-estimated. Another problem with the scheme in Chapter 4 [37] is that it does not take advantage of the application features, because it was intended for *general* real-time applications. In other words, if we only have to support real-time channels with limited, yet important, types — like interactive video — of traffic-generation behaviors and user's performance requirements, we can improve the efficiency and performance of the route-selection scheme significantly, e.g., shorter channel-establishment delay and higher channel-request acceptance rate.

Specifically, in this chapter, we will consider the real-time traffic of interactive video applications. Interactive video applications usually generate frames at some fixed rate[1] and resolution which are both specified according to industry standards. For example, 30 frames per second is the frame rate for live interactive video and the MPEG Video Simulation Model Three (SM3) suggests 352 by 288 pixels per frame for achieving video tape quality [30]. Note that a standardized resolution implies a standardized maximum-frame size. Since video applications of our interest require only a small set of combinations

[1] allowing jitters

of frame-generation rates and maximum-frame sizes, by exchanging and maintaining real-time traffic information among the nodes, the system may be able to prepare for channel establishment even *before* receiving a establishment request. Under this setting, we will develop a scheme which builds and maintains a delay table on each node so that the route-selection problem can be solved by a simple table look-up at the source node [11].

The chapter is organized as follows. Our proposed solution to this problem is presented in Section 5.2. In Section 5.3, we demonstrate via examples the effectiveness of the proposed solution. The chapter concludes with Section 5.4.

## 5.2 The Table-Driven Approach

We first describe the environment and the assumptions under which our table-driven distributed route-selection scheme will be developed. As discussed in Chapter 4, the underlying network is an arbitrary point-to-point network and the generation of real-time messages is assumed to be governed by the linear-bounded model [15]. Based on this message arrival model, as in Chapter 4, we use the scheme proposed in [23,24] to compute the worst-case delay on each link.

In addition to the linear bounded model, we further assume that the number of possible combinations of frame-generation rates and maximum-frame sizes is small, since we are only interested in standardized interactive video applications. Based on the link delay calculated with the delay-estimation method in [23,24] and the above assumptions, we will develop a scheme which builds real-time channel delay tables at each node so that a qualified route may be found by a simple table look-up.

### 5.2.1 Link-Delay Estimation

We will use the same link-delay estimation method as in Chapter 4. Note that this delay-estimation method proposed in [23,24] does *not* include those channels pending for final confirmation in the calculation of MWRT for the new channel-establishment request, but in this chapter, we will treat pending channels differently in two situations. First, as in Chapter 4, during the channel-establishment phase, we will include the load of pending channels in the calculation of MWRTs as if they had already been established. However, the load of pending channels will *not* be included in the real-time delay tables to avoid excessive changes of real-time delay tables, i.e., we do not include pending channels in the

calculation of MWRTs which are used to build real-time delay tables.

As discussed in Chapter 4, including pending channels in the calculation of MWRTs can simplify the channel-establishment phase, but it also causes the over-estimation of MWRT. As will be seen later, because the proposed scheme checks only one route at a time instead of checking all possible routes in parallel (the scheme in Chapter 4 does this), the over-estimation of MWRT is much less likely to occur. Thus, the incorrect rejection decisions due to the over-estimation of MWRT are likely to be made only when there is a very high percentage of real-time traffic so that the over-estimation of MWRT over those *shared* links may make the end-to-end delay larger than the application-required latency.

### 5.2.2 Building Real-Time Delay Tables

Based on the above definition of link delay, we can apply the Bellman-Ford algorithm [9,46] and a loop-free version of the ARPANET's previous routing strategy (APRS) [10,34, 35,39] to build real-time delay tables at each node. As mentioned earlier, the MWRT used to construct real-time delay tables does not include the load of pending channels for two reasons:

- The maximum permissible delay for a link after a final confirmation is likely to be greater than the MWRT computed during the resource-reservation phase. (We will discuss this further in Section 5.2.3.)

- The time between making resource reservation and receiving a final confirmation is usually small, e.g., it could be the time needed for the round-trip from a node on the route under test to the destination.

Thus, if we want to include the load of pending channels in the real-time delay tables, the table entries may have to be modified twice in a short period of time.

When only a small set of standardized combinations of frame-generation rates and maximum-frame sizes needs to be considered, each node in the network can build a loop-free table based on the MWRTs computed according to a pair of maximum-frame size ($S_{max}$ or $C$) and frame-generation rate ($p$). All real-time channels that can be specified by the same pair ($S_{max}$, $p$) are said to be in the same *class*. A node will compute the MWRT on each of its outgoing links as the minimum feasible delay of the corresponding link for *each class* of real-time channels. These MWRTs will be stored in a table, TM, which can be indexed by its neighbors' addresses and has only one field, $r$, for each class of channels,

representing their MWRTs over the corresponding link by considering only those channels already established.

Since the loop-free version of APRS will be used to exchange delay information and maintain real-time delay tables, we will briefly describe the original APRS. In APRS, each node collects and maintains the information about the minimum delays to all other nodes via each of its neighbors. Thus, for every destination–neighbor pair, the information is kept in a 3-tuple form (*destination*, *neighbor*, *delay*). Other information may be needed for the loop-free version of APRS, but we will not discuss this issue here. (See [10,34,35,39] for a detailed account of this.) These 3-tuples are divided into groups based on the destination node. Within a group, they are listed in ascending order of delay. The first entry of each group (the minimum-delay entry to the corresponding destination node) is then used to build a routing table.

Each node periodically exchanges a routing message with its neighbors which contains the node's current routing table. Again, recall that the loop-free version needs to exchange some additional information, and the routing table exchanged may be slightly different from that of the original APRS. After receiving routing messages from its neighbors, each node will update its own routing table based on the information carried in the routing messages and the status of its own outgoing links. As we shall see, the real-time delay table is built in the same way as the routing tables except for the following two differences.

- Only real-time traffic is considered when building real-time delay tables. (APRS's routing tables were built by considering *all* traffic.)
- Real-time delay tables are updated only when a new real-time channel is established or an existing real-time channel is closed. Thus, a node sends "routing" messages to its neighbors only when its set of real-time channels changes. These "routing" messages are called *real-time routing* messages.

Using an example, we will show how to build real-time delay tables.

**Example 5.1:** The class of real-time channels under consideration is specified by $S_{max}$ = 100 Kbps and $p$ = 33 ms. Fig. 5.1 shows the network used for this example. Each link is labeled with a number which represents its transmission speed. Since initially no real-time channels exist, if a channel-establishment request of this class is received, the highest priority will be given to the requested channel. Therefore, the MWRT of this class for each

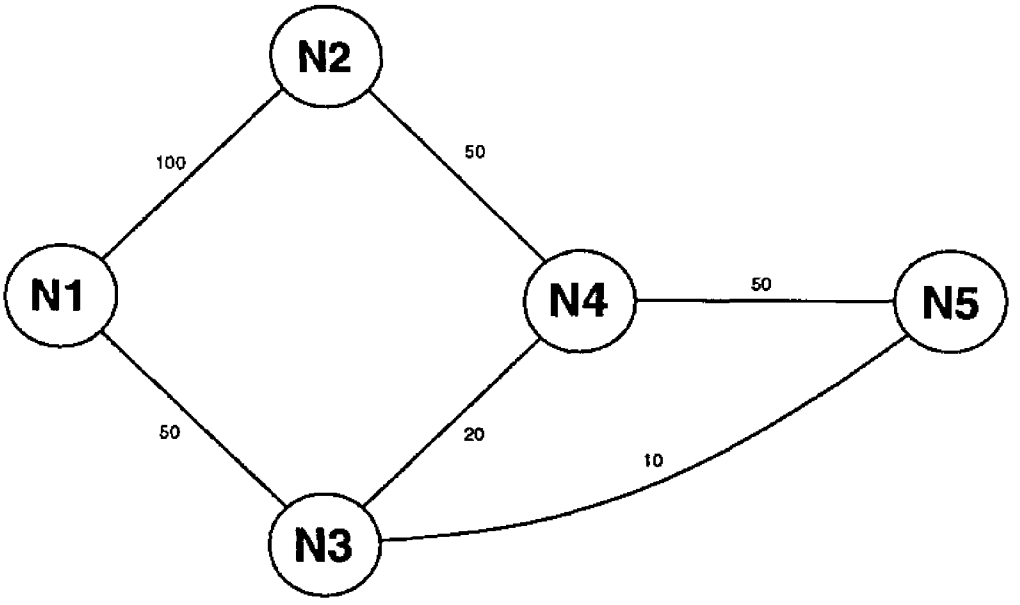



**Figure 5.1:** Example 5.1

link is equal to the maximum service time ($C$) of the class and can be computed as: link N1↔N2: 100/100=1 ms; link N1↔N3: 100/50=2 ms; link N2↔N4: 100/50=2 ms; link N3↔N4: 100/20=5 ms; link N3↔N5: 100/10=10 ms; and link N4↔N5: 100/50=2 ms.

Table 5.1 shows the real-time delay tables for all five nodes in the very beginning. Note that D stands for destination, N stands for neighbor, and d stands for delay in all tables of this chapter. Initially, a node can reach only its neighbors since information about the other nodes is not yet available to the node. The label $\infty$ in an entry represents the case when either the destination cannot be reached via the corresponding neighbor or the path via this neighbor is not loop-free. The least-MWRT path to each destination known so far is used to construct a real-time routing message. Each entry of the message is a 2-tuple, (*destination*, *delay*), where *destination* is not the neighbor to which this message will be sent. For example, the message from N3 to N4 will contain the two entries: (N1,2) and (N5,10).

Since the real-time routing messages are not sent periodically (i.e., the updating procedures of real-time delay tables are not synchronized among nodes), it is not certain what the "next" state of the real-time delay tables will be. However, the "steady-state" real-time delay tables depend only on the currently-anticipated real-time traffic of the established real-time channels. Thus, after all nodes stop sending real-time routing messages (before the next channel establishment or channel closing), the real-time delay tables at that moment can be determined from the current real-time traffic load, regardless of intermediate

| N1's table | | | N2's table | | | N3's table | | | N4's table | | | N5's table | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D | N | d | D | N | d | D | N | d | D | N | d | D | N | d |
| 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 10 |
| 2 | 3 | ∞ | 1 | 4 | ∞ | 1 | 4 | ∞ | 2 | 3 | ∞ | 3 | 4 | ∞ |
| 3 | 3 | 2 | 4 | 4 | 2 | 1 | 5 | ∞ | 2 | 5 | ∞ | 4 | 4 | 2 |
| 3 | 2 | ∞ | 4 | 1 | ∞ | 4 | 4 | 5 | 3 | 3 | 5 | 4 | 3 | ∞ |
| | | | | | | 4 | 1 | ∞ | 3 | 2 | ∞ | | | |
| | | | | | | 4 | 5 | ∞ | 3 | 5 | ∞ | | | |
| | | | | | | 5 | 5 | 10 | 5 | 5 | 2 | | | |
| | | | | | | 5 | 1 | ∞ | 5 | 2 | ∞ | | | |
| | | | | | | 5 | 4 | ∞ | 5 | 3 | ∞ | | | |

**Table 5.1:** Initial real-time delay tables, where D, N, and d stand for destination, neighbor, and delay respectively.

| N1's table | | | N2's table | | | N3's table | | | N4's table | | | N5's table | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D | N | d | D | N | d | D | N | d | D | N | d | D | N | d |
| 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 4 | 7 |
| 2 | 3 | ∞ | 1 | 4 | ∞ | 1 | 4 | ∞ | 2 | 3 | ∞ | 3 | 3 | 10 |
| 3 | 3 | 2 | 4 | 4 | 2 | 1 | 5 | ∞ | 2 | 5 | ∞ | 4 | 4 | 2 |
| 3 | 2 | ∞ | 4 | 1 | ∞ | 4 | 4 | 5 | 3 | 3 | 5 | 4 | 3 | 15 |
| 4 | 2 | 3 | 3 | 1 | 3 | 4 | 5 | 12 | 3 | 5 | 12 | 1 | 3 | 12 |
| 4 | 3 | 7 | 3 | 4 | 7 | 4 | 1 | ∞ | 3 | 2 | ∞ | 1 | 4 | ∞ |
| 5 | 3 | 12 | 5 | 4 | 4 | 5 | 4 | 7 | 5 | 5 | 2 | 2 | 4 | 4 |
| 5 | 2 | ∞ | 5 | 1 | ∞ | 5 | 5 | 10 | 5 | 3 | 15 | 2 | 3 | ∞ |
| | | | | | | 5 | 1 | ∞ | 5 | 2 | ∞ | | | |
| | | | | | | 2 | 1 | 3 | 1 | 2 | 3 | | | |
| | | | | | | 2 | 4 | 7 | 1 | 3 | 7 | | | |
| | | | | | | 2 | 5 | ∞ | 1 | 5 | ∞ | | | |

**Table 5.2:** One possible intermediate state of real-time delay tables.

| N1's table | | | N2's table | | | N3's table | | | N4's table | | | N5's table | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D | N | d | D | N | d | D | N | d | D | N | d | D | N | d |
| 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 4 | 7 |
| 2 | 3 | 9 | 1 | 4 | 9 | 1 | 4 | 8 | 2 | 3 | 8 | 3 | 3 | 10 |
| 3 | 3 | 2 | 4 | 4 | 2 | 1 | 5 | 15 | 2 | 5 | 15 | 4 | 4 | 2 |
| 3 | 2 | 8 | 4 | 1 | 8 | 4 | 4 | 5 | 3 | 3 | 5 | 4 | 3 | 15 |
| 4 | 2 | 3 | 3 | 1 | 3 | 4 | 1 | 5 | 3 | 2 | 5 | 1 | 4 | 5 |
| 4 | 3 | 7 | 3 | 4 | 7 | 4 | 5 | 12 | 3 | 5 | 12 | 1 | 3 | 12 |
| 5 | 2 | 5 | 5 | 4 | 4 | 5 | 4 | 7 | 5 | 5 | 2 | 2 | 4 | 4 |
| 5 | 3 | 9 | 5 | 1 | 10 | 5 | 1 | 7 | 5 | 3 | 15 | 2 | 3 | 13 |
| | | | | | | 5 | 5 | 10 | 5 | 2 | $\infty$ | | | |
| | | | | | | 2 | 1 | 3 | 1 | 2 | 3 | | | |
| | | | | | | 2 | 4 | 7 | 1 | 3 | 7 | | | |
| | | | | | | 2 | 5 | 14 | 1 | 5 | 14 | | | |

**Table 5.3:** Steady-state real-time delay tables.

states.

Table 5.2 represents one possible intermediate state which shows real-time delay tables for all five nodes after receiving one real-time routing message from each neighbor. Table 5.3 shows the steady state of real-time delay tables before the arrival of any real-time channel-establishment request or the termination of any established channel. □

### 5.2.3 The Route-Selection Algorithm

Since the information of all existing channels is necessary for the calculation of a new channel's MWRT as well as for the run-time scheduling of their messages, each node has to maintain another set of tables for the existing channels, one for each outgoing link, in addition to the real-time delay tables. These are called the *tables of existing channels* (TEXCs). Each entry of a TEXC represents a real-time channel which goes through the corresponding link and consists of the following four data fields:

- Channel identifier (ID) which uniquely identifies a real-time channel.
- Class (*class*) of the channel.
- Status (*status*) of the channel, established (1) or pending (0).
- The maximum permissible link delay ($d$) for the channel.

As discussed in Example 4.2, in order to be consistent with the way channel priorities are assigned for the link-delay estimation [23, 24], these entries are placed in ascending order of $d$ values.

When the source node wishes to establish a real-time channel to another node, say B, it will try to find the current least-MWRT route by considering the traffic of all existing channels. Recall that only those already-established channels are figured in real-time delay tables. Thus, the source node will send a real-time channel-request message (*Req*) to the next node on the least-MWRT route, which contains a channel ID, the destination address (*dest*), the channel class (*class*), the end-to-end delay bound $D$, the path (*path*) and the total number of hops (*hops*) this message has traveled thus far, and the accumulated delay $d^a$. Initially, the $d^a$ field is set to the MWRT of the corresponding outgoing link, *path* is set to the source node, and *hops* is set to 1. As in Chapter 4, *hops* is included in the request message only for convenience of presentation.

```
Procedure rcv_req
If (Req.dest = A) then {reply_req; return;}
for (i = 1 to number_of_entry(RTDT[Req.dest]) {
        If (Req.d^a + RTDT[Req.dest][i].d) > Req.D) then {send_reply(reject); return;}
        nextnode := RTDT[Req.dest][i].N;
        If (no pending channel in TEXC[nextnode]) then {
          insert_req(TM[nextnode]); forward_req(nextnode); return;
        }
        ;;; compute new MWRT, including established and pending channels.
        r := compute_MWRT(nextnode, 1);
        If (Req.d^a + RTDT[Req.dest][i].d - TM[nextnode] + r ≤ Req.D) then {
          insert_req(r); forward_req(nextnode); return;
        }
}
send_reply(reject);
```

**Figure 5.2:** Procedure of processing a channel-establishment request.

Fig. 5.2 describes the procedure of handling a channel-establishment request after node A receives the request. This procedure can also be applied to the source node by setting $d^a := 0$, *hops* := 0, and *path* to an empty string. In order to find the next node of a qualified route, Procedure *rcv_req* uses the destination, B, as an index to the real-time delay table and searches through all routes whose delays to B are not greater than the *remaining* user-required delay bound ($Req.D - Req.d^a$). The **for** loop and the first **if** statement in the loop serve as this function. Since the entries in the real-time delay tables are placed in ascending

**Procedure** *insert_req*($r$)

$TEXC.ID := Req.ID$;
$TEXC(Req.ID).class := Req.class$;
$TEXC(Req.ID).status := pending$;
$TEXC(Req.ID).d := r$;

**Procedure** *forward_req*(*nextnode*)

$Req.d^a := Req.d^a + TEXC(Req.ID).d$;
$Req.hops := Req.hops + 1$;
;;; concatenate A and *Req.path*.
$Req.path := A \cdot Req.path$;
;;; all other fields remain the same.
forward this request message to *nextnode*.

**Figure 5.3:** Procedures of inserting and forwarding a request.

order of delays to the destination, the search starts from the first entry and can terminate if the current entry cannot satisfy the **if** statement.

After passing the first **if** statement in the loop, if there is no pending channel (the second **if** in the loop), this entry is selected and appropriate actions will be taken by calling Procedure *insert_req* and *forward_req*. Otherwise, we have to re-compute the MWRT ($= r$) for the corresponding outgoing link, because the real-time delay tables do not account for the pending channels. If the increase of MWRT due to the pending channel ($-TM[nextnode] + r$ part in the third **if** statement) doesn't make the delay to the destination greater than $Req.D - Req.d^a$, this entry can be selected as the channel's route. The maximum permissible delay on this link ($d$ field in TEXC) is set to $r$, instead of obtaining it directly from $TM[nextnode]$. TM is a table of MWRTs (in Section 5.2.2), and RTDT (in Procedure *rcv_req*) is used to denote real-time delay tables.

Fig. 5.3 describes the procedures of inserting a new (pending) channel to TEXC and forwarding a request. As can be seen from these procedures, most fields are directly copied from the establishment-request message to TEXC and the forwarding message.

Fig. 5.4 shows the operations that a destination will perform after receiving a channel-establishment request. Since the $d^a$ field of a channel-establishment request represents the sum of MWRTs of all links on the path from the source to destination, the user-specified end-to-end delay bound, $D$, may be larger than $d^a$, i.e., we are allowed to spend more time than the corresponding MWRT when sending a message over each intermediate link.

**Procedure** *reply_req*

**If** (*the application accepts the request*) **then** *send_reply(accept)*;
**else** *send_reply(reject)*;

**Procedure** *send_reply(accept)*

*nextnode* := *head*(*Req.path*);
*Reply.ID* := *Req.ID*;
*Reply.flag* := *accept*;
*Reply.diff* := (*Req.D* − *Req.d*$^a$)/*Req.hops*;
*Reply.path* := *tail*(*Req.path*);
send *Reply* to *nextnode*.

**Figure 5.4:** Procedure of processing a channel-establishment request at the destination.

In such a case, the authors of [23,24,37] proposed that $D - d^a$ could be divided evenly into *hops* parts by the destination node and distributed to all links along the path. The deadline of a real-time message of this particular channel over an intermediate link is the channel's MWRT of that link plus $(D - d^a)/hops$. Note that one may also choose to divide $D - d^a$ in proportion to each link's MWRT, i.e., the maximum permissible delay over link $\ell$ is computed as $(D - d^a) \times \frac{MWRT_\ell}{d^a} + MWRT_\ell$, where $MWRT_\ell$ is the MWRT of link $\ell$. However, since this method may make the link-delay deadline unnecessarily small over a link which has small MWRT we will adopt the method proposed in [23,24,37].

Since this link deadline is stored in the table of existing channels ($d$ field in TEXC) and used for run-time scheduling, $(D - d^a)/hops$ is included in the channel-establishment confirmation message (by Procedure *send_reply(accept)*) from the destination to source via the same path the corresponding request message had traveled (but in the opposite direction). Let *Reply* denote a channel-establishment confirmation message which consists of four fields: $ID$, $flag$ (accept or reject), $diff$ ($= (D - d^a)/hops$) and *path* (the remaining path back to the source node). Fig. 5.4 describes how a positive confirmation message is constructed (*send_reply(accept)*), and Fig. 5.5 shows the operations the intermediate nodes will perform when receiving a (positive or negative) reply message (*forward_reply*). Note that *head*(*list*) represents the first element of *list*, and *tail*(*list*) represents the remaining list after removing *head*(*list*) from *list*.

The operations in Procedure *update*(*node*) of Fig. 5.6 are necessary to keep these real-time delay tables and TMs up-to-date after a new channel is established or an existing channel is torn down. Basically, nodes which receive a positive reply to a channel-establishment

**Procedure** *forward_reply*

**If** ($Reply.flag = reject$) **then** delete the entry $TEXC(Reply.ID)$;
**else** {
  $TEXC(Reply.ID).status := established$;
  $TEXC(Reply.ID).d := TEXC(Reply.ID).d + Reply.diff$;
  insert this entry in the ascending order of $d$ field.
  ;;; update real-time delay tables if necessary
  ;;; assume the message is sent/forwarded by node N.
  update(N);
}
$nextnode := head(Reply.path)$;
$Reply.path := tail(Reply.path)$;
forward this reply message to the next upstream node *nextnode*.

**Figure 5.5:** Procedure of handling reply messages.

request will re-compute the MWRT by considering only those established channels already (including the one just accepted). Based on this new MWRT, real-time routing tables and TM are updated and a new real-time routing message is generated and sent to all neighbor nodes. When a node receives a real-time routing message from a neighbor node, it will follow the procedure described in Section 5.2.2 to update its real-time delay tables.

The operations necessary to keep real-time delay tables and TMs up-to-date during the channel-disconnect phase are straightforward. We require one of the two communicating peers to send a disconnect message through the route of the real-time channel to the other communicating peer. In the disconnect message, only $ID$ needs to be included. All intermediate nodes and the source node will delete the corresponding entry from their TEXCs and call *update* (Fig. 5.6) to update both TMs and real-time delay tables. Real-time routing messages may also be sent as discussed in Section 5.2.2. In the next section, we will present examples to illustrate the operations of establishing a channel.

## 5.3 Examples

We will use the network in Fig. 5.1 for the demonstrative examples in this section.

**Example 5.2:** In Example 5.1, we constructed the real-time delay tables for a class of real-time channels specified by $S_{max}$ = 100 Kbytes and $p$ = 33 ms. In this example, we

```
Procedure update(node)
message := φ;
for each class class of real-time channels {
        ;;; compute the new MWRT, including only established channels.
        r := compute_MWRT(node, 0);
        If (r ≠ TM[node]) then {
          update_RTDT(node, r);
          TM[node] := r;
          table := U_{i=all destinations}{(i, RTDT[i][1].d)};
        }
        message := message ∪ {(class, table)};
}
send a real-time routing message, message, to all neighbor nodes;
Procedure update_RTDT(node, r)
for all destinations, dest {
        i := 1;
        while (i ≤ number_of_neighbors) {
          If (RTDT[dest][i].N ≠ node) then i := i + 1;
          else {RTDT[dest][i].d := RTDT[dest][i].d + r - TM[node]; break; }
        }
}
```

**Figure 5.6:** Procedure of updating real-time delay tables.

want to establish a channel (ID = 1:1) of this class (class 1) with an end-to-end delay bound $D$ = 32 ms from N1 to N5.

**Link-bandwidth reservation:** N1 will use N5 as the index to its real-time delay table (Table 5.3), and find the next node, N2, on the least-MWRT path to N5 with MWRT = 5 ms. Since $D$ > 5 ms and there is no pending channel, N1 will insert this channel into its TEXC for link N1→N2. By looking up the table of MWRTs (TM[N2]=1) for this class, the entry representing this channel in TEXC can be set to (1:1,1,0,1). The channel-establishment request Req(1:1,N5,1,32,N1,1,1) can also be sent to the next node, N2.

After receiving this request message, N2 will call *rcv_req* to handle this request message and forward it to the next hop on the least-MWRT path to N5. As can be seen from

Table 5.3, N4 is the next hop and the delay is 4 ms. Since the accumulated delay ($d^a$) carried in the request is 1 ms, i.e., the end-to-end MWRT ($1 + 4 = 5$ ms) is not greater than $D = 32$ ms and there is no pending channel on link N2→N4, this channel request will be inserted into N2's TEXC for link N2→N4. Because TM[N4]=2 (for class 1), the entry in TEXC will be set to (1:1,1,0,2). A request message, Req(1:1,N5,1,32,N2N1,2,3), will also be forwarded to N4.

N4's operations are similar to N2's. The entry inserted in N4's TEXC (link N4→N5) for this request is (1:1,1,0,2), and the request message, Req(1:1,N5,1,32,N4N2N1,3,5), is forwarded to N5.

N5's operations will be different from those of N2 and N4, because N5 is the destination of the requested channel. If the peer application at N5 decides to accept this channel request, a positive reply message will be constructed and sent back to N1 via the same path of the request message traveled but in the opposite direction. Since $D = 32$ ms is greater than the accumulated delay ($d^a = 5$ ms) carried in the request message, the difference, $D - d^a$, will be divided evenly into *hops* parts and distributed to all links along the path. Thus, using Procedure *send_reply(accept)* (Fig. 4.3), $diff = (32-5)/3 = 9$ and the positive reply message will be Reply(1:1,accept,9,N2N1). This reply message will then be sent to N4 (the head of path carried in the request message).

**Channel-acceptance confirmation:** After receiving the positive reply from N5, N4 will call *forward_reply* and update the entry in TEXC (link N4→N5) representing channel 1:1 to (1:1,1,1,11), where the third component indicates this channel to have been established, and the fourth component ($2 + 9 = 11$) shows the maximum permissible delay of this channel over link N4→N5. This positive reply (after removing *head*(*Reply.path*)) will then be forwarded to the next upstream node (N2) specified by *head*(*Reply.path*).

Due to the establishment of channel 1:1, the MWRTs of channels (of all classes) over link N4→N5 have to be re-computed. However, by using the link estimation method described in Section 5.2.1, the MWRT for class-1 channels still remains to be 2 ms. Thus, N4's real-time delay tables will remain unchanged and no real-time routing messages will be sent. Note that although the MWRT of class-1 channels does not change, the MWRTs for other classes may change after channel 1:1 is established. Both TMs and real-time delay tables for these classes will be updated and real-time routing messages will also be sent to reflect the new MWRTs of these classes. We will demonstrate the update operations in Example 5.3.

| N1's table | | N2's table | | N3's table | | N4's table | | N5's table | |
|---|---|---|---|---|---|---|---|---|---|
| N | MWRT | N | MWRT | N | MWRT | N | MWRT | N | MWRT |
| 2 | 3 | 1 | 3 | 1 | 6 | 2 | 6 | 3 | 30 |
| 3 | 6 | 4 | 6 | 4 | 15 | 3 | 15 | 4 | 6 |
| | | | | 5 | 30 | 5 | 6 | | |

**Table 5.4:** Tables of MWRTs where N stands for neighbor.

The operations performed by N2 and N1 are similar to N4's. The TEXC for link N2→N4 will contain an entry (1:1,1,1,11) and the TEXC for link N1→N2 will contain an entry (1:1,1,1,10). All real-time delay tables and TMs remain unchanged because the establishment of channel 1:1 does not increase the MWRT of class-1 channels over the links of the route.

If the application at the destination node N5 refuses to accept this request, a negative reply will be sent back via the same path. After receiving the negative reply, all nodes will delete the corresponding entry from their TEXCs and forward the reply message to the next upstream node specified by the *path* in the reply message. □

**Example 5.3**: Another class (class 2) of real-time channels specified by $S_{max}$ = 300 Kbytes and $p$ = 20 ms will be used in this example. Table 5.4 shows the tables of MWRTs (TMs) for all five nodes after channel 1:1 is established (Example 5.2), and Table 5.5 shows the steady state of real-time delay tables for class-2 channels. We will establish a class-2 channel (ID = 1:2, destination = N5, and $D$ = 30 ms) and then show the change in both TMs and real-time delay tables after its establishment.

As in Example 5.2, the path N1N2N4N5 will be chosen because it is the least-MWRT path from N1 to N5. Thus, a request will be sent from N1 to N5 via this path and each node on this path will insert a corresponding entry into its own TEXC to reflect the existence of channel 1:2. The entries to be inserted into the TEXCs of N1, N2 and N4 are: link N1→N2: (1:2,2,0,3), link N2→N4: (1:2,2,0,6), and link N4→N5: (1:2,2,0,6). If the peer application at the destination node accepts this channel-establishment request, N5 will compute $diff = 30 - (3 + 6 + 6) = 5$ ms and send a positive Reply(1:2,1,5,N2N1) to N4.

After receiving the positive reply to the request for establishing channel 1:2, N4 will set the status of this channel to "established" and the maximum permissible delay to 6+5 = 11 ms in the TEXC (link N4→N5). Then, N4 will re-compute the MWRT of both classes over link N4→N5, including only the load of established channels. For class-2 channels, TM[N5]

| N1's table | | | N2's table | | | N3's table | | | N4's table | | | N5's table | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D | N | d | D | N | d | D | N | d | D | N | d | D | N | d |
| 2 | 2 | 3 | 1 | 1 | 3 | 1 | 1 | 6 | 2 | 2 | 6 | 3 | 4 | 21 |
| 2 | 3 | 27 | 1 | 4 | 27 | 1 | 4 | 24 | 2 | 3 | 24 | 3 | 3 | 30 |
| 3 | 3 | 6 | 4 | 4 | 6 | 1 | 5 | 45 | 2 | 5 | 45 | 4 | 4 | 6 |
| 3 | 2 | 24 | 4 | 1 | 24 | 4 | 4 | 15 | 3 | 3 | 15 | 4 | 3 | 45 |
| 4 | 2 | 9 | 3 | 1 | 9 | 4 | 1 | 15 | 3 | 2 | 15 | 1 | 4 | 15 |
| 4 | 3 | 21 | 3 | 4 | 21 | 4 | 5 | 36 | 3 | 5 | 36 | 1 | 3 | 36 |
| 5 | 2 | 15 | 5 | 4 | 12 | 5 | 4 | 21 | 5 | 5 | 6 | 2 | 4 | 12 |
| 5 | 3 | 27 | 5 | 1 | 30 | 5 | 1 | 21 | 5 | 3 | 45 | 2 | 3 | 39 |
| | | | | | | 5 | 5 | 30 | 5 | 2 | ∞ | | | |
| | | | | | | 2 | 1 | 9 | 1 | 2 | 9 | | | |
| | | | | | | 2 | 4 | 21 | 1 | 3 | 21 | | | |
| | | | | | | 2 | 5 | 42 | 1 | 5 | 42 | | | |

**Table 5.5:** Steady-state real-time delay tables for class-2 channels after establishing channel 1:1, where D, N, and d stand for destination, neighbor and delay respectively

is then modified to 14, because any new channel request of class 2 has to be given priority lower than both channel 1:1 and 1:2 in order to retain the performance guarantees of existing channels. However, the MWRT of class-1 channels does not change. Thus, the real-time delay tables (Table 5.5) will be updated and real-time routing messages containing only the class-2 information will be sent to N4's neighbors. For example, the message sent to N2 will contain three entries: (N5,14), (N3,15), (N1,21), and the message sent to N3 will also contain three entries: (N5,14), (N2,6), and (N1,9). Note that the third entry in the message to N2 is the result of loop-free routing. Since the least-MWRT path from N4 to N1 is N4N2N1 which includes N2, the second best path N4N3N1 is sent to N2 as the (loop-free) least-MWRT path.

The positive reply will then be forwarded to N2 then to N1. The operations of N2 are similar to those of N4. The MWRTs of both classes over link N2→N4 have to be re-computed and the real-time delay tables have to be updated accordingly. The MWRT of class 2 over link N2→N4 will increase to 14 and that of class 1 remains unchanged. Thus, {(N5,28),(N4,14),(N3,29)} will be sent to N1, and {(N1,3),(N3,9),(N5,38)} will be sent to N4 as the real-time routing messages for class-2 channels.

After receiving the confirmation for establishing channel 1:2, N1 will also re-compute the MWRTs of both classes. Since both MWRTs do not change over link N1→N2, no further actions are necessary.

In addition to the positive reply messages, the real-time routing messages generated

| N1's table | | | N2's table | | | N3's table | | | N4's table | | | N5's table | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D | N | d | D | N | d | D | N | d | D | N | d | D | N | d |
| 2 | 2 | 3 | 1 | 1 | 3 | 1 | 1 | 6 | 2 | 2 | 6 | 3 | 4 | 21 |
| 2 | 3 | 27 | 1 | 4 | 35 | 1 | 4 | 24 | 2 | 3 | 24 | 3 | 3 | 30 |
| 3 | 3 | 6 | 4 | 4 | 14 | 1 | 5 | 45 | 2 | 5 | 53 | 4 | 4 | 6 |
| 3 | 2 | 32 | 4 | 1 | 24 | 4 | 4 | 15 | 3 | 3 | 15 | 4 | 3 | 45 |
| 4 | 2 | 17 | 3 | 1 | 9 | 4 | 1 | 23 | 3 | 2 | 15 | 1 | 4 | 15 |
| 4 | 3 | 21 | 3 | 4 | 29 | 4 | 5 | 36 | 3 | 5 | 44 | 1 | 3 | 36 |
| 5 | 2 | 31 | 5 | 4 | 28 | 5 | 4 | 29 | 5 | 5 | 14 | 2 | 4 | 12 |
| 5 | 3 | 35 | 5 | 1 | 38 | 5 | 5 | 30 | 5 | 3 | 45 | 2 | 3 | 39 |
| | | | | | | 5 | 1 | 37 | 5 | 2 | ∞ | | | |
| | | | | | | 2 | 1 | 9 | 1 | 2 | 9 | | | |
| | | | | | | 2 | 4 | 21 | 1 | 3 | 21 | | | |
| | | | | | | 2 | 5 | 42 | 1 | 5 | 50 | | | |

**Table 5.6:** Steady-state real-time delay tables for class-2 channels after establishing channel 1:1 and 1:2.

by N4 and N2 will also be used to update real-time routing tables by those nodes which receive them as discussed in Section 5.2.2. Following the procedure of building real-time delay tables, Table 5.6 shows the steady-state real-time delay tables for all five nodes after channel 1:2 is established.

## 5.4 Conclusion

In this chapter, we have proposed a table-driven distributed route-selection scheme which is guaranteed to find a qualified route, if any, for each real-time channel-establishment request. By equipping a real-time delay table with each node, the proposed scheme can choose a route for each real-time channel requested by a table look-up.

# CHAPTER 6

# AN APPLICATION: FIELDBUS

## 6.1 Introduction

In this chapter, we will discuss the real-time communication issues of the FieldBus protocol which is a new industrial standard, and apply the schemes discussed thus far to an automated factory environment which uses FieldBus. An automated factory (AF) is usually composed of several workcells (or simply cells), each of which contains robots, sensors, and transport mechanisms. A multiaccess bus connects all devices in a workcell. A bridge is used to connect two or more workcells. Hence, the network of an AF consists of many links, each of which can be viewed as a multiaccess bus. The ability to provide predictable inter-process communication is of great significance to an AF, because unpredictable communication delays may lead to missing the deadline of one or more communicating tasks. In this chapter, we will address the issue of providing real-time communication on FieldBus which is designed to support time-critical communication to and from devices in manufacturing systems.

Due to the nature of manufacturing systems, most time-critical communication is likely to occur between devices in the same workcell, and hence, a fast connection establishment procedure is desirable. Since the physical area of a workcell is usually small and there are often many devices (nodes) in a workcell, a multiaccess network is a natural candidate for connecting devices in a workcell. On the other hand, since the ability to provide predictable communication between any two devices in different workcells is also essential to the system, we will focus on developing a "fast" real-time channel establishment scheme within a workcell (i.e., two nodes on the same multiaccess bus), while providing the ability to support real-time communication between any two workcells.

The integrated scheme presented in this dissertation is an ideal solution for providing

real-time communication between devices in a manufacturing system. The statistical real-time channels on multiaccess networks can be used to provide real-time communication between devices in a workcell. For devices in different workcells, the proposed distributed route-selection schemes can be used to establish inter-cell real-time channels so as to provide inter-cell real-time communication services with performance guarantees. Note that although both proposed route-selection schemes are applicable in this case, we will use the table-driven scheme to show that our scheme can be integrated into the FieldBus protocol.

The chapter is organized as follows. In Section 6.2, we provide a brief overview of the data-link layer of the FieldBus protocol. The proposed scheme is discussed in Section 6.3. Section 6.4 discusses the compatibility between the proposed scheme and the FieldBus protocol. The performance evaluation of the proposed scheme is presented in Section 6.5, and the chapter concludes with Section 6.6.

## 6.2 Overview of FieldBus

Our main goal is to analyze and enhance the data-link layer of FieldBus protocol. Although most of the data-link layer protocol of FieldBus has been developed well, the support for time-critical message communication is not completely specified. To facilitate our presentation, we need to briefly describe the FieldBus data-link layer.

The FieldBus data-link layer is designed to support time-critical message communication for manufacturing systems and process controls. The FieldBus network is composed of several links, each of which is a multiaccess bus connecting all the devices in the workcell; thus, the entire FieldBus network is a collection of multiaccess buses which are further connected via several bridges. The data-link layer protocol of the FieldBus is similar to the well-known Token Bus protocol, except that in the FieldBus protocol, there is a link control entity which is responsible for token allocation of the corresponding link. Although FieldBus is designed to support time-critical communication, it also provides support for non-real-time data transmission.

Since the primary goal of the FieldBus is to support real-time communication, several changes (to the OSI model) have been made to improve the performance. One important performance measure of real-time communication is the worst-case delay in sending a message. In order to reduce the delay in sending a message, unlike the OSI model, FieldBus has only three layers: Physical Layer, Data Link Layer and Application Layer. Among

the three layers, the Data Link Layer is the most important, since the Physical Layer is just an interface of the physical medium, and the Application Layer is user-dependent, i.e., beyond our control. The Data Link Layer is modeled as a series of four sub-layers: Link Access, Link Scheduling, Multi-link and Data Link Connection (DLC) Management, and Bridge Management. As far as the real-time communication issues within a workcell are concerned, it is sufficient to consider only the Link Access and the Link Scheduling sub-layers.

Before describing the FieldBus protocol, we need to formally define a "data link entity." In the data-link layer of FieldBus, a **Data Link Entity** (DLE) is a logically active object, such as a copy of the executing program, which is able to send/receive packets to and from the interconnection network and acts according to the data-link layer protocol of FieldBus. Therefore, there could be several DLEs on a node which is physically attached to the interconnection network, such as computers, sensors, or any manufacturing devices. However, it is conceptually simple to treat each DLE as a node. Thus, the terms "DLE" and "node" will be used interchangeably.

The data-link layer protocol of FieldBus is a delegated token protocol. There is a local control entity on each multiaccess link which is responsible for scheduling messages on the local link, i.e., this local control entity is responsible for managing the token. This local control entity allocates a token to other entities based on their needs and each token grants the receiver entity a certain amount of time for transmitting data on the shared medium.

The data-link layer service of FieldBus provides both connectionless and connection-oriented communication between two peer communicating DLEs. The connection-oriented communication service, which supports both real-time and non-real-time communication, requires the source DLE to establish a connection first, thus allowing the source node and intermediate nodes to collect and exchange the information needed for the delivery of packets, e.g., buffer requirements and route information. In general, a static routing strategy is used for the delivery of connection-oriented packets, although it is not required by the standard [3,5,6]. If a real-time connection is requested, the system has also to reserve sufficient bandwidth and perform appropriate admission tests in the connection-establishment procedure.

The connectionless service, which supports only non-real-time communication, is similar to the traditional multiaccess packet switching networks. However, connectionless communication is allowed only between two DLEs on the same link in the current draft proposal

[3,5,6]. This limitation might be relaxed later, because there is no good reason to establish a connection via several bridges for exchanging only a few non-real-time packets. Since the connection-establishment process is relatively time-consuming, connectionless communication should have been allowed for non-real-time messages across link boundaries.

There are four classes of DLEs in the FieldBus data-link layer: Basic, Link Master (LM), Link Active Scheduler (LAS), and Bridge. Basic and LM classes are conceptually the same, except that the LM class DLEs are equipped with more functions, while the Basic class DLEs have only those functions which are absolutely necessary for adequate operation on a FieldBus network. In general, these two classes of DLEs are the "user" nodes on the FieldBus networks. For simplicity, the term "DLEs" or "nodes" will be used to mean both classes in the rest of this chapter.

Unlike other popular timed-token protocols (e.g., token rings, token buses, FDDI), FieldBus has a *local* control unit, LAS DLE, for each Fieldbus link. (Note that a FieldBus network is composed of a set of links.) The LAS DLE is responsible for scheduling messages on the local link. It receives, and responds to, scheduling requests from all DLEs on the same link by giving a token to one of these DLEs which then assumes the exclusive right to use the link over some time period specified in the token. A bridge DLE is one which performs a store-and-forward function to connect two or more separate multiaccess links. In the draft proposal of the FieldBus data-link layer protocol [3,5,6], the routing strategy used by bridges is not specified.

All normal communication requests for use of a link are scheduled by the LAS of the link. The LAS generates "polling" tokens for DLEs on the link, and the receiver DLE responds immediately by returning a message which may include requests for future scheduling and the priorities of the current requests. The LAS derives a schedule according to some scheduling policy and provides the token to the "winner" DLE.

Based on the above brief description of FieldBus protocol and the nature of workcells in an AF, one can see that the LAS is equivalent to the LCU in our scheme. Since most real-time communication is likely to take place between two peer DLEs which are located on the same link, most time-critical communication can be handled by the local LAS, i.e., the LCU. In the rest of this chapter, we will use LAS and LCU interchangeably.

For handling time-critical communication between devices in different workcells, a distributed scheme is implicitly assumed in the draft proposal of FieldBus protocol [3,5,6], since the existence of a global network manager is not mentioned at all. In order to provide

predictable communication for real-time applications, the system has to select a "qualified" route, then reserve certain resources (e.g., link bandwidth) along the selected route, verify certain conditions are met (e.g., the user-specified end-to-end delivery delay bound), and guarantee these conditions will always be met during the lifetime of the application. Our distributed route-selection schemes for establishing multi-hop real-time channels are ideal for handling such inter-cell time-critical communication.

## 6.3 Basic Approach

As we discussed in the schemes covered in the previous chapters, non-real-time traffic load is ignored when dealing with real-time traffic because real-time traffic is given higher priority. Thus, we will focus on real-time traffic and ignore non-real-time traffic in the rest of this chapter.

Because time-critical communication is most likely to take place between devices within a workcell, the real-time communication problem associated with the FieldBus protocol can be decomposed into two related sub-problems. The first sub-problem is to provide real-time communication between two peer DLEs which are located on the same link. The second sub-problem deals with the ability to establish real-time channels between two peer DLEs which are located on two different links. Since the first sub-problem is more likely to happen than the second one, the channel establishment procedure and the resource management for the channels of two peer DLEs on the same link should be designed efficiently. For the second sub-problem, we can apply the proposed distributed route-selection scheme to solve the inter-cell time-critical communication problems.

The communication capacity of each link is divided into two parts. One part is concerned with intra-link communication which is managed by the local LAS (i.e., LCU). The other part deals with inter-link communication which is managed by a table-driven distributed route-selection scheme (Chapter 5). In other words, all intra-workcell communication between two peer DLEs on a link are scheduled by the LAS on that link. The inter-workcell real-time communication between two peer DLEs on two different links are managed by the distributed route-selection algorithm, and scheduled by the LAS of each link of the path over which the corresponding channel runs. The fraction of the link capacity assigned to each of these two parts depends on the distribution of communication demand on each link. That is, the LAS of a link can reserve a different fraction of link capacity for local

(or intra-link) communication based on the characteristics of local communication demand. Since communication traffic may vary, the reserved capacity may also vary.

The chief advantage of dividing the link capacity is that we can have a fast channel establishment procedure for intra-cell channels without triggering the expensive global operations for keeping the route-selection tables up-to-date. Therefore, the route-selection tables need not to be changed due to the addition/deletion of intra-cell real-time channels, i.e., the route-selection tables for inter-cell real-time channels will be more stable. The proposed solutions to the two sub-problems are described in detail in the following two subsections.

### 6.3.1 Intra-link Communication

Intra-link (intra-cell) communication occurs between two peer DLEs on the same multiaccess link. This type of communication can be either connection-oriented or connectionless. Since real-time communication requires a bounded delay in message delivery, we need to reserve all the required resources in advance in order to guarantee that all messages can be delivered before their deadlines. This implies that real-time communication must be connection-oriented. We will use the schemes proposed in Chapter 2 and 3 to provide the intra-cell real-time communication. By considering the LAS equivalent to the LCU in the proposed schemes, intra-link real-time communication will be handled by the LAS DLE on the corresponding multiaccess link, because the LAS is designed to be the local control entity for scheduling messages on that local link under the FieldBus protocol [3,5,6].

The operations of the channel establishment procedure in Chapter 2 and 3 for the peer DLEs are the same as those in the description of the draft proposal [3,5,6]. That is, the source DLE will make a channel establishment request which includes necessary information for establishing the channel according to the FieldBus protocol, such as a frame control field, the destination address, and the quality of service. In order to support real-time communication, the traffic pattern and the resources requirement also need to be specified in a channel establishment request message. That is, in addition to the required information in the draft proposal, each node needs to compute a MTRT (maximum token return time) and RTHT (real-time token holding time) and include them in the channel establishment request message which is sent to the LAS.

In addition to the operations specified in the draft proposal, the LAS must respond to all channel establishment requests for local real-time channels since all real-time channels

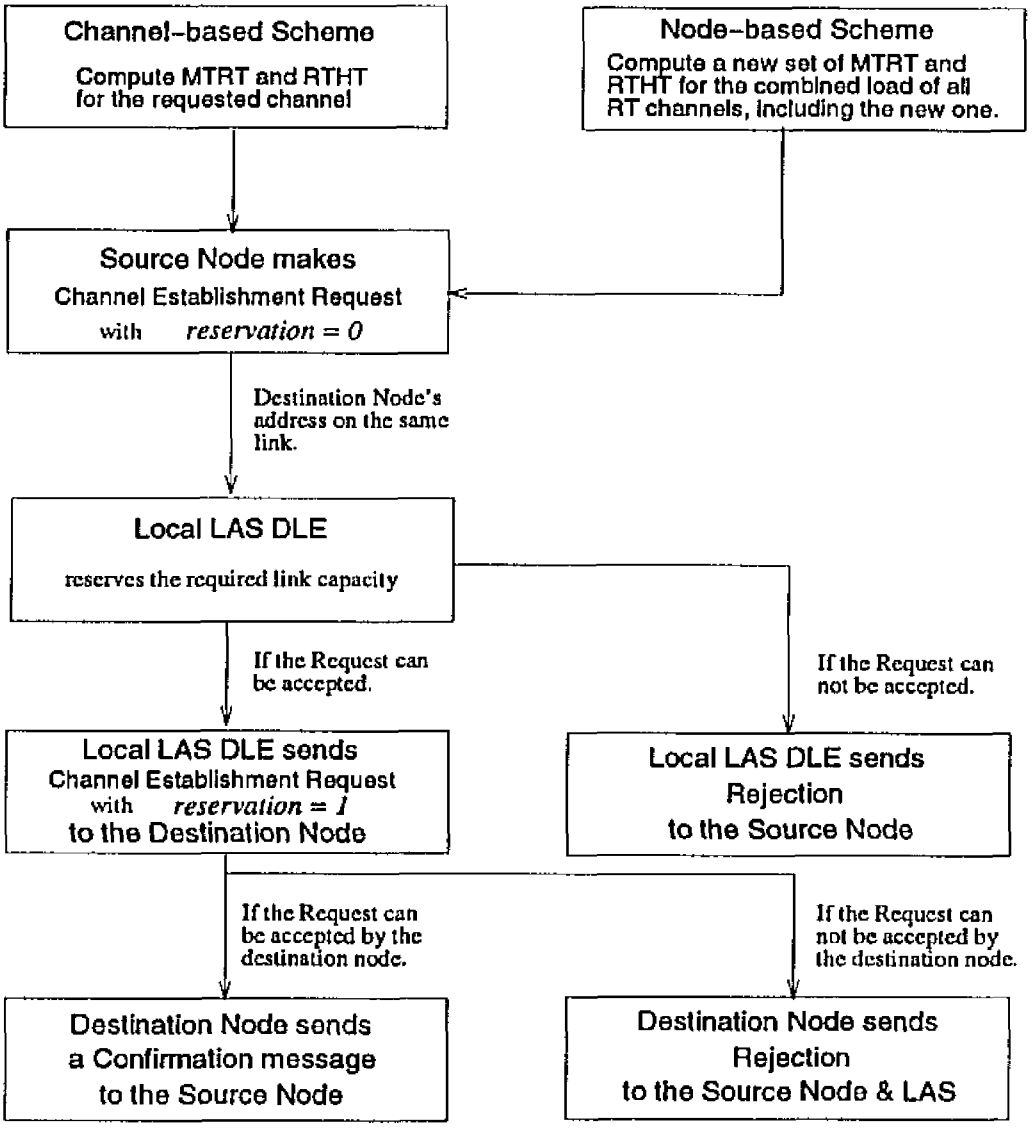



**Figure 6.1:** Procedure for handling an intra-link channel establishment request.

running through this local link are subject to its approval. Note that the LAS's approval is not required for non-real-time channels since no resource has to be reserved in advance for them. A channel establishment request message includes a *reservation* (0 or 1) field, representing whether the local LAS has reserved the requested capacity or not. When the LAS receives a real-time channel establishment request from a local DLE with a destination DLE located on the same link, the LAS tries to reserve the requested link capacity and respond to this channel establishment request. Consequently, in addition to the computation of the corresponding set of MTRT and RTHT after adding this new channel, a real-time channel establishment request is handled in three steps (see Fig. 6.1), whereas a non-real-time channel establishment request requires only two steps. First, the requesting DLE sends

a channel establishment request to the corresponding local LAS with $reservation = 0$ which indicates that the channel establishment request has not yet been approved by the local LAS. Both the LAS and the designated receiver node receive this request, but the designated receiver node will ignore all real-time channel establishment requests with $reservation = 0$. The LAS will respond to this request by sending out the modified channel establishment request message with the requested quality of service and $reservation = 1$, if the LAS can reserve a sufficient capacity and accept this channel establishment request. Otherwise, the LAS will reject this request by sending a rejection message to the requesting DLE. After receiving a positive response from the LAS, the receiver DLE will respond as described in the draft proposal, i.e., report the channel establishment request to the destination user, and the user will decide whether to accept this request or not. If the channel establishment request cannot be accepted, the receiver DLE will respond with a disconnection message, which will be received simultaneously by both the LAS and the requesting DLE since the link is a multiaccess bus. The LAS will release all the resources reserved for this channel.

If there are already too many real-time channels established which have almost exhausted all the link capacity under the control of a LAS, the LAS may try to reserve more link capacity for local usage. However, this is subject to the availability of the remaining link capacity under the management of the distributed route-selection scheme.

In order to be compatible with the current draft proposal, if the source DLE requires an immediate reply for a real-time channel establishment request, the receiver DLE will respond with an acknowledgement immediately. Otherwise, the source node will send the channel establishment request again as specified in the draft proposal. If the addressed destination still does not respond, the source DLE will report the failure of the destination DLE to the user, and the LAS will stop the reservation process and free all the resources reserved thus far for this particular request. When an immediate reply is required, the LAS will respond to the requesting node in the very next time slot after detecting the required immediate reply.

### 6.3.2 Inter-link Communication

The portion of link bandwidth which is allocated for inter-cell real-time communication is under the control of one (or more) bridge(s) (FieldBus terminology) which is physically attached to the link. If there is more than one bridge involved in the management of inter-cell real-time communication for nodes in the corresponding cell, the portion of link

bandwidth which is allocated for inter-cell real-time communication will be divided further and assigned to these involved bridges. Note that in order to prevent fragmentation of link bandwidth, the number of such bridges should be limited. Therefore, each bridge has complete control of a certain portion of link bandwidth which will be allocated to this bridge on a time-division-multiplexing (TDM) basis on the shared medium. From the viewpoint of this bridge, it becomes the LCU (or LAS) of the link with a reduced link speed (because of TDM); thus, it may use this portion of capacity to send data to any node on this link or allow some node on this link to transmit data. Consequently, the bridge becomes an end node of this corresponding link (with a reduced transmission speed) which behaves like a link in a point-to-point network. Under this environment, the table-driven distributed route-selection scheme proposed in Chapter 5 can be used to establish inter-cell real-time channels.

The entire operation for establishing a channel between two DLEs on different links is conceptually similar to that when they are on the same link. The only significant difference is that the real-time channel establishment requests are granted by all the bridges along the route which is chosen by the route-selection algorithm, rather than a local LAS. The real-time channel establishment procedure still consists of three steps (see Fig. 6.2). First, the requesting DLE sends a real-time channel establishment request to a bridge on the link. In addition to the parameters specified in the FieldBus protocol, three parameters, $S$ (maximum message size), $p$ (minimum message inter-arrival time) and $D$ (user's end-to-end delay bound requirement), must be included in the channel establishment request message. Note that the request can also be made without an "addressed" bridge. In this case, all bridges attached to the link will use their real-time delay tables to determine which bridge has the shortest-MWRT-delay-path to the destination. And this bridge (with the shortest-MWRT-delay-path to the destination) will become the "addressed" bridge.

Since the destination DLE is not on the same local link (as the requesting DLE), the local LAS will ignore this channel establishment request, and the addressed bridge will reserve sufficient resources (link bandwidth) and forward this request to the next bridge along the route which is selected according to the real-time delay tables. (Note that in the non-real-time channel establishment request case, the bridge will also forward the request to the addressed destination DLE, without reserving resources). If no qualified path can be found according to the scheme proposed in Chapter 5, a rejection message will be sent to the source node and all resources reserved so far will be released.

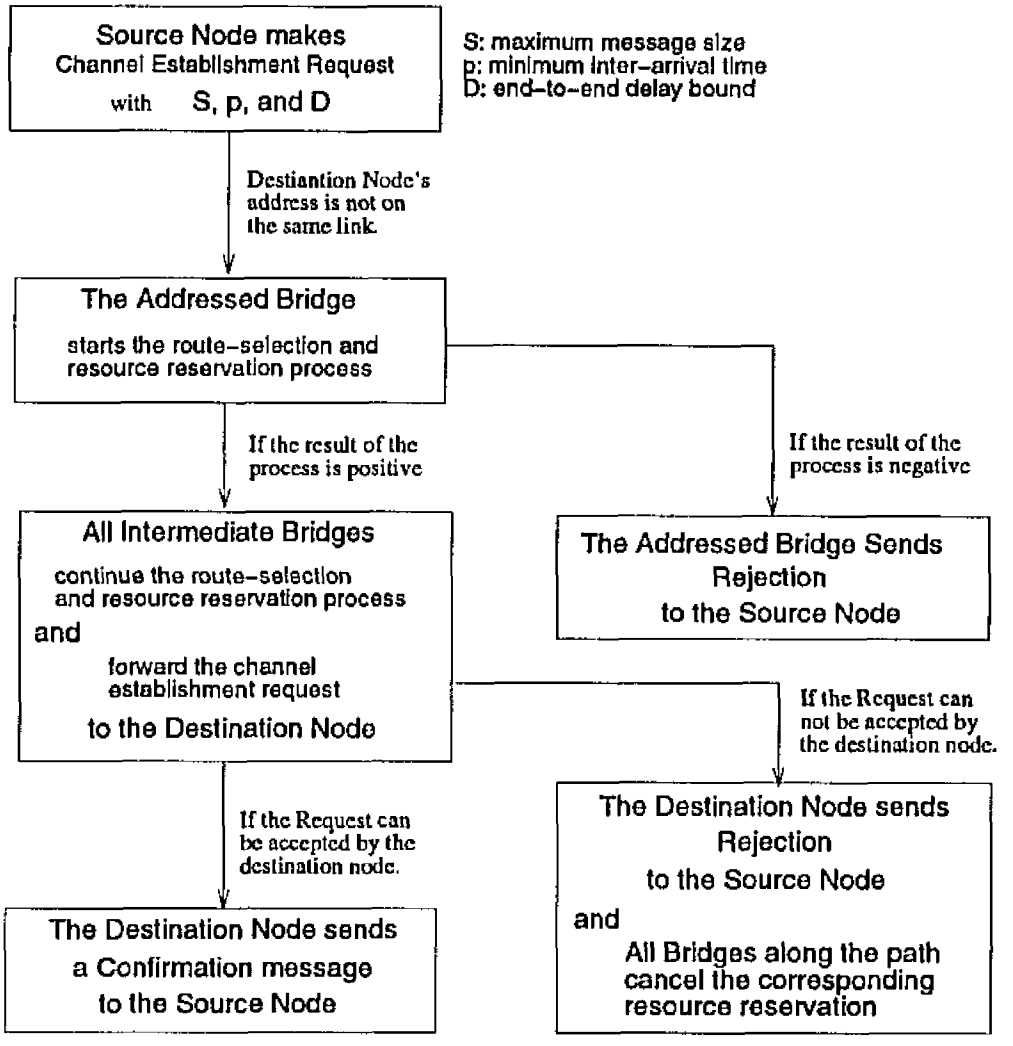



**Figure 6.2:** Procedure for handling an inter-link channel establishment request.

Finally, after the destination DLE receives the channel establishment request, the receiver DLE will report this request to the destination user who will then decide its acceptance/rejection. In case of acceptance, the destination DLE will send a confirmation message back to the requesting DLE along the same path (but in the opposite direction) along which the request message traveled. All the operations which are necessary to maintain the integrity of the real-time delay tables are performed according to the scheme in Chapter 5. Otherwise, a disconnection message will be sent along the established path to the source node, thus making all intermediate bridges aware of this disconnection so that all reserved resources for this real-time channel can be released.

If the destination DLE does not exist or respond to the channel establishment request within a timeout period, the failure of the destination DLE can be detected by the bridge

on the same link where the destination DLE resides and will be reported to the source node by returning the channel establishment request to the source node. The source node may choose to retry or inform the requesting application.

## 6.4 Compatibility with the FieldBus Protocol

Since most parts of the FieldBus protocol have been well developed and gained general acceptance from the manufacturing and process control communities, the proposed scheme for real-time communication must be compatible with the FieldBus protocol. The most notable aspects of the proposed scheme are the introduction of the real-time delay tables for route-selection algorithm, the division of link capacity, and the difference in establishing real-time and non-real-time channels.

Although the FieldBus protocol does not specify how the inter-cell real-time communication should be handled, it does imply this problem should be solved in a distributed manner. Therefore, using a distributed route-selection scheme to solve the inter-cell real-time communication problem is compatible with the FieldBus protocol.

The second difference introduced by the proposed scheme is the division of the link capacity into two parts which are controlled by either the corresponding local LAS or the distributed route-selection scheme. This link capacity division can easily be accommodated into the current protocol. In the current FieldBus protocol, the entire link communication capacity is controlled by the local LAS, regardless of whether the communication is intra-link or inter-link and whether it is connectionless or connection-oriented. In order to improve the utilization and traffic balancing of the network, we introduced the table-driven distributed route-selection scheme for inter-link real-time communication, and as mentioned before, a portion of link capacity is controlled by the route-selection scheme (bridges). Each LAS can still function as described in the FieldBus protocol, except that some portion of link capacity is assumed to have already been reserved for global usage. The LAS just follows the instruction (in the request form) of bridges (the result of route-selection algorithm) when assigning the "global-usage" portion of link capacity to some designated bridge DLE(s) and/or the application DLE(s). Since bridges are only allowed to allocate a pre-negotiated portion of the link capacity, and never exceed it, the LAS must follow the bridge's instruction. At the same time, the LAS may also grant its local requests without any further checking with bridges. The portion of link capacity which is controlled by each

bridge can be negotiated, i.e., when a bridge becomes active, it first negotiates with the LAS of this local link for a portion of link capacity. Then this bridge starts to build the real-time delay tables and broadcasts real-time delay messages based on this portion of link capacity. When this bridge (based on the result of route-selection scheme) decides to grant an inter-cell real-time channel request, it will inform the local LAS to allocate certain portion of link capacity (out of the part of capacity which is under control of this bridge) to the source node of the channel. Note that only the bridge on the same link as the source node needs to inform the LAS of that link to allocate tokens to the source node. Other bridges do not need such operations, because they receive data from another bridge which has its own portion of link capacity. After a channel is closed, the amount of capacity which is allocated for the source node is returned to the control of the corresponding bridge.

From a scheduling perspective, the local LAS schedules the tokens according to the requests granted by both itself and the bridges. Since both the LAS and the bridges cannot exceed their pre-negotiated limits, the messages for real-time channels can always be delivered in time once the channel has been established.

The real-time channel establishment procedure is also different from the non-real-time counterpart. Establishing a real-time channel requires three steps, while establishing a non-real-time channel requires only two steps. This difference comes from the fact that a real-time channel establishment has to be granted by either the local LAS (intra-cell) or bridges (inter-cell) based on the result of the route-selection algorithm. In the inter-cell cases, the request messages will not reach the destination node before the appropriate resource reservation has been made; thus, the procedure of handling inter-cell real-time channel establishment requests is the same as that of the non-real-time channel.

However, in the intra-cell cases, the real-time channel establishment request is considered valid by the node where the destination DLE resides even before the local LAS reserves resources for the request. In such a case, the destination DLE may receive not-yet-accepted ($reservation = 0$) request messages, but it will ignore them. So, the real-time channel establishment request can be processed by the DLEs in the same way as a non-real-time channel request. The FieldBus protocol uses a state-driven procedure to manage a non-real-time channel, which is briefly described here. As can be seen later, the establishment of a real-time channel can also be achieved by this state-driven procedure. The source (requesting) node makes a real-time or non-real-time channel request, initiates a state machine for the channel, and enters CR-sent (Connection Request has been sent) state [3, 5, 6]. A

node in CR-sent state that has received a channel establishment confirmation enters DATA state and begins transmission. On the other hand, a node receives a non-real-time or a valid real-time channel establishment request will create a state machine associated with this request and enter the CR-rcvd state. After receiving the user's positive response to this request, the node in CR-rcvd state sends out a channel establishment confirmation to the requesting node, enters DATA state, and begins transmission. A user's negative response (reject or close) to a state machine will force the node to send a disconnection message and terminate the associated state machine. Therefore, real-time channel establishment requests can be processed in the same state-driven model with minor modification (adding a *reservation* bit) as the non-real-time channel case.

Although the real-time channels can be established by using the same procedure as the non-real-time channel at individual nodes, the bridges and the LAS have to be equipped with the ability to handle real-time channel requests, make appropriate reservations, and respond to such requests adequately. The bridge DLEs also require additional functions to make correct run-time scheduling and flow control in order to provide predictable communication [22,23]. Although these changes to the LAS and bridge are non-trivial, as mentioned before, they are compatible with the draft FieldBus protocol, and are necessary in order to support real-time communication.

## 6.5 Simulation

In this section, we present a numerical example to show the performance of the proposed modification of FieldBus. We use a typical manufacturing system topology (see Fig. 6.3) as the interconnecting network for the simulation. The goal is to derive the likelihood of accepting real-time channel establishment requests and the non-real-time traffic throughput under different link load conditions for both intra-cell and inter-cell communication [3,5,6, 22,29]. In Fig. 6.3, each cell represents a workcell in an AF and there is a LAS in each cell. The $LAS_0$ represents the LAS for the backbone link.

Real-time communication in manufacturing systems is usually periodic, and the deadline of each message is related to the period of the message. For example, the controller of a workcell in a manufacturing system usually must read sensors in this workcell periodically, and these sensor data should be collected and processed before the next period begins. In this example, each real-time channel has its own period, and in each period, a fixed amount

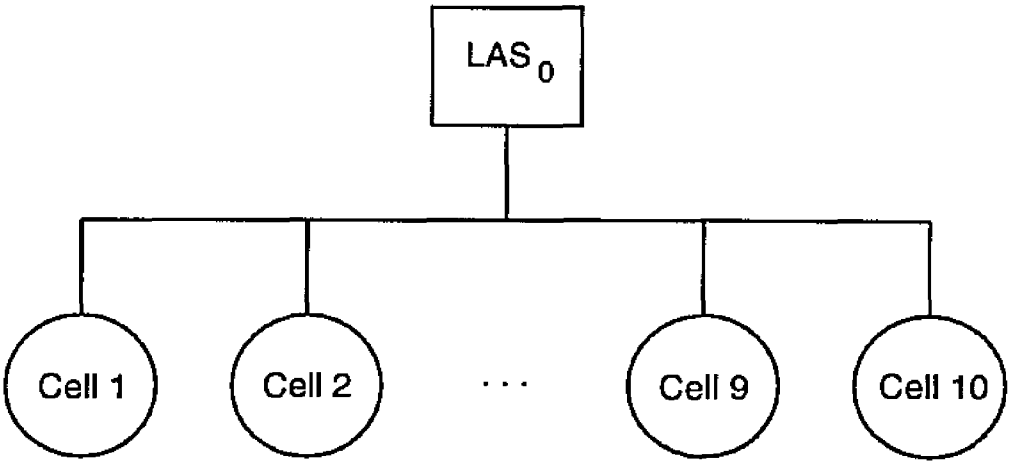



**Figure 6.3:** Two-level hierarchical network for an AF

of time is assigned to the source node if the channel can be established, because the size of each time-critical message is limited in the FieldBus and each message may potentially include the maximum number of bytes. This fixed amount of time is assumed to be used for handling a message of 256 bytes — which includes a maximum of 128 bytes user data for time-critical messages — and all overheads, such as token passing time, transmission delay, and framing overheads. Connections which need to send more than 256 bytes in a period are simulated by making multiple channel requests.

Real-time channel request arrivals are assumed to be exponentially distributed with a fixed rate. The period of each channel is assumed to be uniformly distributed within the range from 20 ms to 500 ms based on the nature of the manufacturing system under consideration. In addition to the arrival rate and period, the lifetime of each channel is assumed to be normally distributed with a mean ranging from 1 second to tens of seconds and a small variance, since a channel is requested by a certain entity only for a limited lifetime. For example, a robot may need a channel with a short-life device for 10 seconds when it operates on an assembly line, but it may only need a channel with a long-life device for the next several seconds when the transport belt is moving. Therefore, it is reasonable to assume that the lifetime of each kind of channel is normally distributed with a small variance and a fixed mean. In this example, we use several different lifetime distributions, but, as long as the total requested load remains the same, the likelihood of accepting real-time channel establishment requests and the average maximum achievable non-real-time message throughput do not make any significant difference.

The following figures show the probability of accepting an intra-cell/inter-cell real-time

channel request under a wide range of requested real-time channel loads. When the requested real-time channel load is under 100% of the link capacity, the speed of the link is important to the acceptance probability. The reason for this tendency can be given as follows. The requested channel load of a link is the sum of the loads of all requested channels regardless of whether they are accepted or not. Insofar as a single channel request is concerned, the load of a single real-time channel request is measured by the number of *slots* (each of which is the time required to handle a 256-byte packet) the channel needs per second or its period in terms of slots, and this load occupies a higher percentage of the capacity of a low speed link than it does on a high speed link. Therefore, when the requested real-time channel load is less than 100% of the link capacity (i.e., the link has some unused capacity), the probability that a new requested channel cannot be accommodated in the remaining capacity of a low speed link is much higher than a high speed link.

Figs. 6.4 and 6.5 show, respectively, the likelihood of accepting real-time channel establishment requests and the average maximum achievable non-real-time message throughput for intra-cell communication. As can be seen in Fig. 6.4, the higher the link speed, the higher the acceptance probability. The non-real-time message throughput has the opposite trend. In Fig. 6.5, the low speed link has a higher non-real-time message throughput in terms of percentage of the link capacity, since a smaller percentage of link capacity is reserved for real-time communication.

Figs. 6.6 and 6.7 show, respectively, the likelihood of accepting real-time channel establishment requests and the average maximum achievable non-real-time message throughput for inter-cell communication. In this example, we use a 4 Mbps backbone to connect 10 workcells, and three different link capacities for channels within a workcell, i.e., 128 Kbps, 256 Kbps, and 512 Kbps. In our strategy, inter-cell and intra-cell communications do not interfere with each other, since they use different portions of the link capacity. These two figures follow the same trend as Fig. 6.4 and Fig. 6.5. However, the percentage of real-time channel establishment requests accepted drops much faster for inter-cell channel requests, since an inter-cell channel can be established only if all links on the path of the channel have sufficient capacity to support this channel, while an intra-cell channel only needs one link which has sufficient capacity to establish. Because a smaller percentage of link capacity is reserved for real-time channels, the average maximum achievable non-real-time message throughput is higher in the inter-cell communication case (as can be seen from Figs. 6.5 and 6.7).

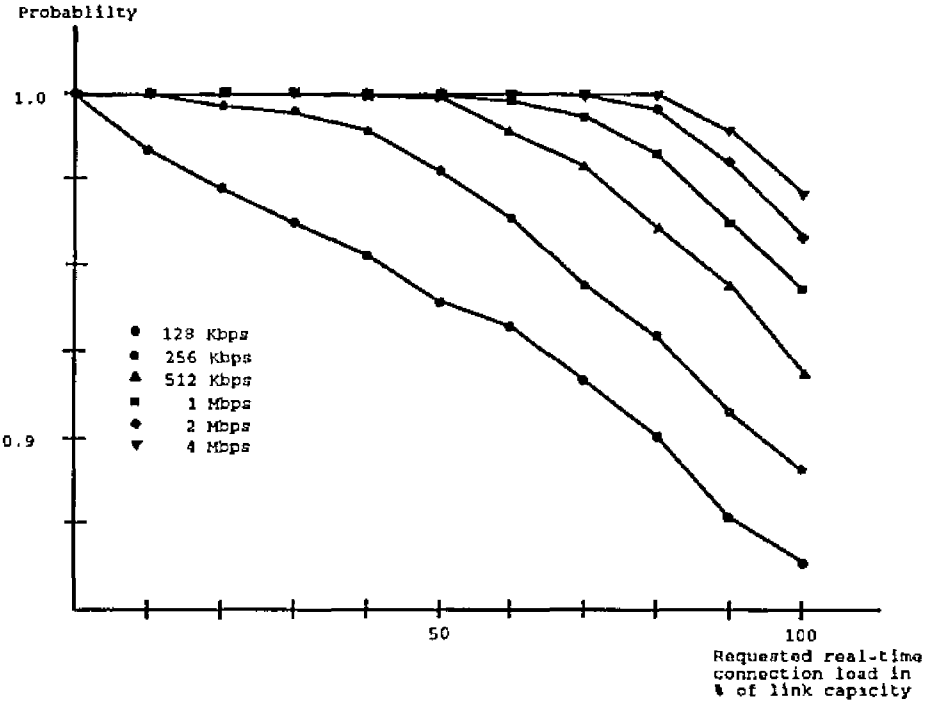



**Figure 6.4:** Probability for honoring intra-cell real-time channel requests

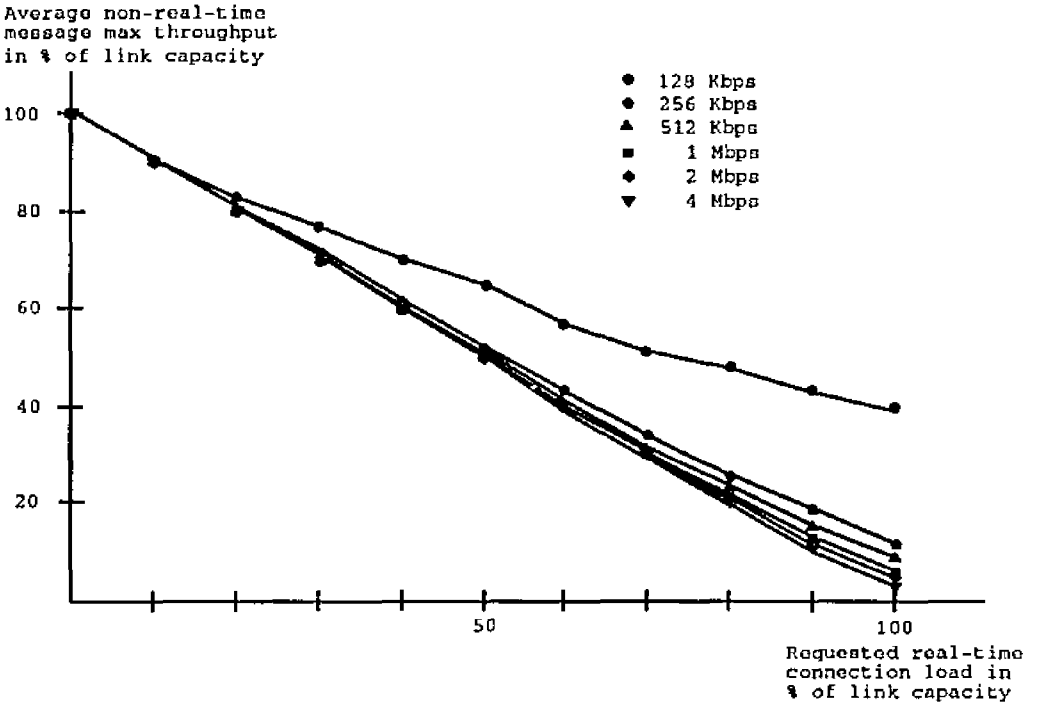



**Figure 6.5:** Average maximum throughput for intra-cell non-real-time messages

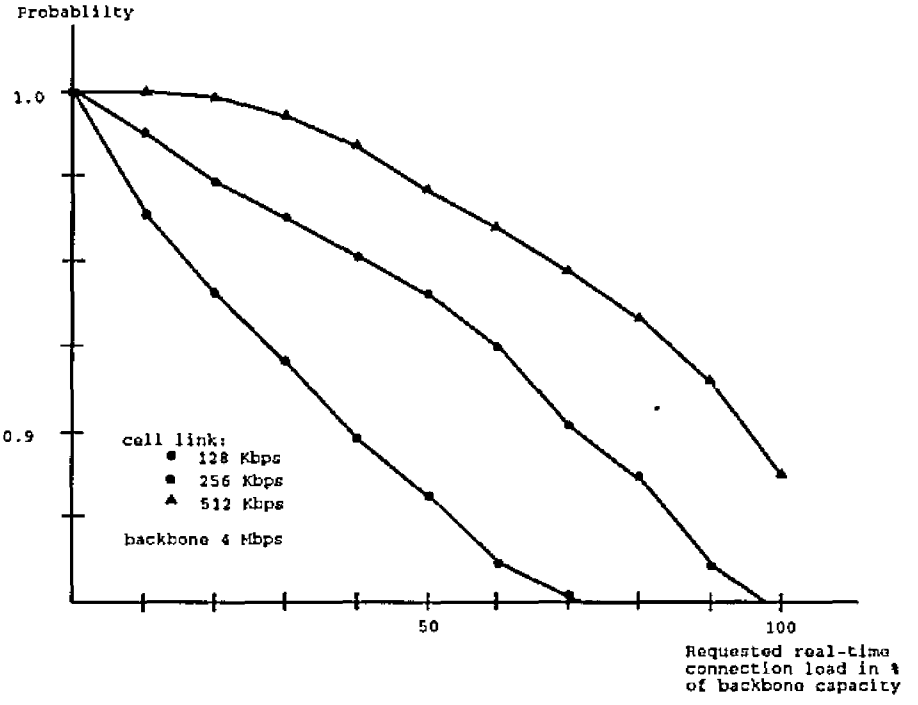



**Figure 6.6:** Probability for honoring inter-cell real-time channel request

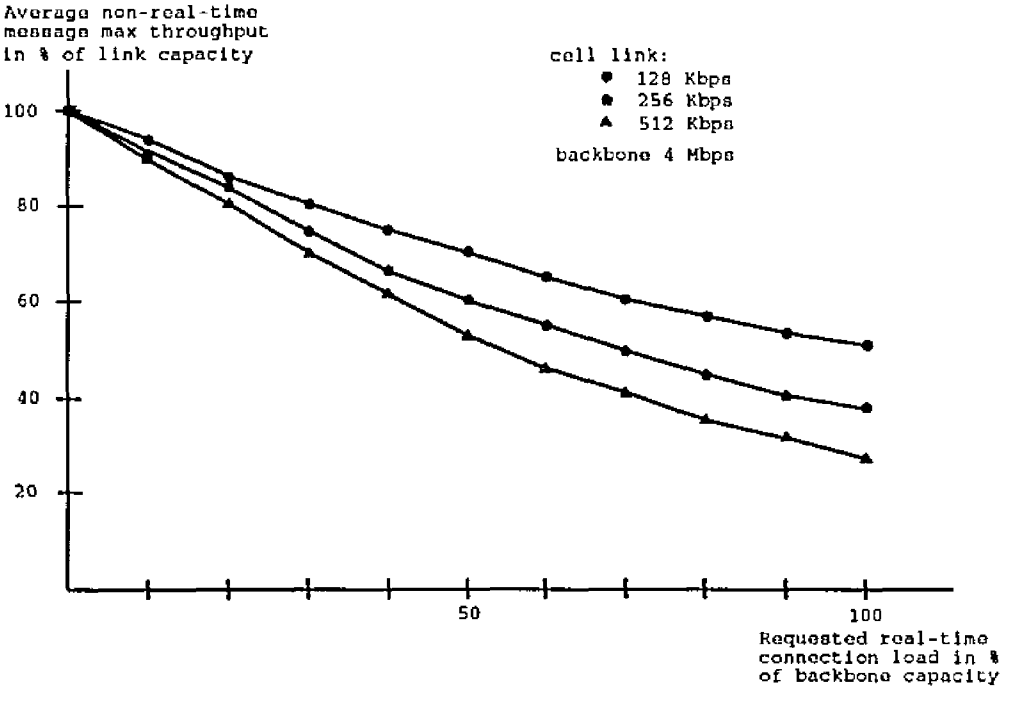



**Figure 6.7:** Average maximum throughput for inter-cell non-real-time messages

## 6.6 Conclusion

In this chapter, we proposed a strategy to handle real-time communication under the FieldBus protocol which provides end-to-end delivery delay guarantees for time-critical messages. This strategy provides a fast local mechanism for establishing intra-workcell real-time channels, while supporting global inter-workcell real-time channels. The proposed strategy is fully compatible with the current draft proposal of FieldBus protocol, and also provides flexibility for the choice of scheduling algorithms, adaptability for different traffic loads. Numerical examples are also given based on a typical manufacturing network.

# CHAPTER 7

# SUMMARY AND FUTURE WORK

## 7.1 Summary of the Contributions

This dissertation has treated a new and increasingly important subject, real-time communication with statistical performance guarantees. The main contributions are summarized as follows.

**Channel-based statistical real-time channels:** The channel-based scheme developed in Chapter 2 can provide real-time communication services with statistical and absolute performance guarantees for multiaccess network environments. Since no complex scheduling algorithm is required at the node level, the channel-based scheme can be implemented on a very simple node which may have only little computing power. In addition to the ability to provide performance guarantees, the proposed scheme can also improve network utilization by using statistical (as opposed to hard) real-time channels. Our simulation results have also shown that the proposed scheme is effective and efficient in supporting both real-time and non-real-time communication.

**Channel-multiplexing strategy:** This strategy takes advantage of multiplexing real-time channels originating from the same source node to reduce the link capacity that needs to be reserved and thus improve the overall network utilization. In addition to improving of network utilization, this scheme can also preserve the capability of independent addition/deletion of real-time channels and can provide performance guarantees.

**Multiple-Due-Date (MDD) scheduling algorithm:** The MDD algorithm proposed in Section 3.3 can solve the frame-dependency problem which is common in many video applications. By combining MDD with channel-multiplexing, the integrated scheme

can significantly improve the network utilization and, at the same time, provide performance guarantees even in the presence of frame-dependency. Our simulation (motion video) results have shown that the integrated scheme can reduce the link capacity that needs to be reserved to the level of average real-time traffic from the original worst-case level of traffic. This reduction is practically important since the capacity reserved in the worst case is often significantly larger than that in an average case.

**Distributed route-selection algorithm:** The route-selection strategy studied in Chapter 4 can guarantee to find a "qualified" route, if any, satisfying the performance requirement of the requested channel without compromising any of the existing guarantees. The proposed scheme can also eliminate the common reliability/performance bottleneck of a centralized route-selection scheme, while improving efficiency over the centralized and other distributed schemes. Although the proposed solution starts with searching all possible routes *in parallel*, it prunes infeasible routes quickly and its worst-case operational overhead is shown to be only a linear function of the number of links in the network.

**Table-driven distributed route-selection scheme:** This scheme is designed to support real-time channels with limited, yet important, types — like interactive video — of traffic-generation behaviors. By supporting only limited types of traffic, we can improve the efficiency and performance of the route-selection scheme significantly. The proposed scheme uses the Bellman-Ford shortest path algorithm to build real-time delay tables, and hence, can solve the route-selection problem by a simple table look-up.

**FieldBus application:** FieldBus is a new industrial standard which is designed to support time-critical communication in process control and manufacturing systems. In Chapter 6, we show that our real-time channel scheme can be used with the FieldBus protocol to enhance the capability of providing real-time communication in manufacturing systems.

## 7.2 Future Work

In this dissertation, we have studied the problem of providing real-time communication with statistical performance guarantees. However, there are still many unsolved problems

and issues associated with real-time communication with performance guarantees. The problems listed below are some of them.

**Fault-tolerance of real-time channels:** Fault-tolerance is an important issue for real-time communication. If nodes/links on the route of a real-time channel fail, the system has to be able to recover quickly from the faults. Conventionally, we either establish multiple routes in advance or try to find an alternate route after the failure. Thus, the system will suffer either low network utilization due to over-reservation or a long recovery latency because of the channel establishment upon detection of each failure. By using real-time delay tables, we may be able to develop a scheme which can choose a qualified alternate route using a simple table look-up.

**Dynamic routing of real-time channels:** The route-selection problem we have studied so far is based on the assumption that the network topology is static, i.e., does not change unless a failure occurs. However, if one of the communication peers is a moving object, such as a vehicle in a cellular communication environment, the static assumption does not hold. Therefore, a new route-selection policy must be developed to support such an environment.

**Implementation issues:** In this dissertation, although we have presented most of our approaches in the form of procedures or flow-charts and provided many simulation results, we still placed more emphasis on analytical rather than implementation-oriented issues. It is important to have thorough knowledge of implementation-related issues. For example, we need to know how to obtain and also improve the approximation of the "typical" input traffic distribution through experience and experiments. Experiment-based studies may help us achieve this goal.

**Applications:** Since traffic specifications and performance requirements vary from application to application, it is not feasible to define a universal optimal network protocol and hardware structure for all applications. Therefore, the integration of the proposed scheme with various applications and the necessary modification due to the requirements of specific applications are important to the utility of our schemes.

# APPENDIX A

## List of Symbols and Acronyms

| | | |
|---|---|---|
| $LAN$ | : | Local area network. |
| $MDD$ | : | Multiple-due-date scheduling algorithm. |
| $EDD$ | : | Earliest-due-date first (deadline-driven) scheduling algorithm. |
| $D$ | : | User-specified delay bound. |
| $Z$ | : | User-specified performance-related probability. |
| $S_{max}$ | : | Maximum packet size in Chapter 2 and 3 or maximum message size in Chapter 4 and 5. |
| $B_{max}$ | : | Maximum burst size. |
| $G_{max}$ | : | Maximum packet-arrival rate. |
| $TTRT$ | : | Target token rotation time (FDDI). |
| $THT$ | : | Token holding time. |
| $MTRT$ | : | Maximum token return time for a real-time channel. |
| $RTHT$ | : | Real-time token holding time for a real-time channel. |
| $LCU$ | : | Link control unit. |
| $M$ | : | Maximum number of packets that can be generated by a real-time channel in an interval of length D. |
| $P_{max}$ | : | Transmission time for a maximum-size packet, i.e., packet time. |
| $N_{max}$ | : | Maximum number of packets that a real-time channel can transmit during each token allocation. |
| $G$ | : | Average packet arrival rate of a real-time channel. |
| $RBU$ | : | Reserved-but-unused link capacity. |
| $MTRT_N$ | : | Maximum token return time for node N. |
| $RTHT_N$ | : | Real-time token holding time for node N. |
| $MTRT^n$ | : | Maximum token return time for channel $n$. |
| $RTHT^n$ | : | Real-time token holding time for channel $n$. |
| $N$ | : | Random variable representing the number of packet arrivals for a real-time channel within one MTRT. |
| $X$ | : | Random variable representing the number of average packet arrivals for a real-time channel within one packet time, i.e., $X = \frac{N}{MTRT}$. |

| | | |
|---|---|---|
| $R$ | : | Random variable representing the RBU link capacity per packet time. |
| $Y$ | : | $Y = R - X$. |
| $R_{max}$ | : | Maximum message generation rate. |
| $C_i (C_i)$ | : | Maximum time required to transmit a message of a real-time channel ($i$), i.e., maximum service time of a message of a real-time channel ($i$) on a link. |
| $p_i (p_i)$ | : | Minimum message inter-arrival time of a real-time channel ($i$). |
| $I^i_{min}$ | : | $p_i$. |
| $d_i (d_i)$ | : | Link (delay) deadline for a real-time channel ($i$), i.e., maximum permissible delay for a real-time channel ($i$) on a link. |
| $MWRT$ | : | Minimum worst-case response time. |
| $TEC$ | : | Table of established channels. |
| $ID$ | : | Unique identifier for a real-time channel. |
| $TPR$ | : | Table of pending requests. |
| $d^a$ | : | Accumulated MWRT delay from the source node to the current node. |
| $timeout$ | : | Expiration time of a real-time channel establishment request. |
| $r$ | : | MWRT of a link. |
| $Req$ | : | Real-time channel establishment request message. |
| $path$ | : | The path from the source node to the current node. |
| $hops$ | : | Total number of hops that a request message has traveled thus far. |
| $diff$ | : | $(D - d^a)/hops$. |
| $LPR$ | : | List of already-processed requests. |
| $Reply$ | : | Reply message of a real-time channel establishment request. |
| $APRS$ | : | ARPANET's previous routing strategy. |
| $TM$ | : | Table of MWRTs. |
| $TEXC$ | : | Table of existing channels. |
| $RTDT$ | : | Real-time delay table. |
| $AF$ | : | Automated factory. |
| $DLE$ | : | Data-link entity. |
| $LAS$ | : | Link active scheduler. |
| $LM$ | : | Link master. |

# BIBLIOGRAPHY


[1] *FDDI token ring media access control.* ANSI Standard, 1988.

[2] *FDDI hybrid ring control.* Draft Proposal to American National Standard, 1989.

[3] *Industrial Automation Systems — Systems Integration and Communications — Field-Bus (draft version 7) (ISA SP50 1991).* Instrument Society of America, 1991.

[4] *Identifying user requirements for systems supporting time-critical communications.* Technical Report for TCCA Rapporteurs' Group of ISO/TC184/SC5/WG2, March 1992.

[5] *Industrial Automation Systems — Systems Integration and Communications — Field-Bus (draft version 10) (ISA SP50 1992).* Instrument Society of America, 1992.

[6] *Industrial Automation Systems — Systems Integration and Communications — Field-Bus (draft version 11) (ISA SP50 1992).* Instrument Society of America, 1992.

[7] D. P. Anderson, S. Y. Tzou, R. Wahbe, R. Govindan, and M. Andrews, "Support for continuous media in the dash system," *Proc. 10-th Int'l. Conf. on Distributed Computing Systems*, pp. 54–61, May 1990.

[8] K. R. Baker, *Introduction to Sequencing and Scheduling*, Wiley, New York, 1974.

[9] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall International, 1987.

[10] C. C. Cheng, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves, "A distributed loop-free routing algorithm suitable for arbitrary link weights," Technical Report CSS-89-05, Dept. Electrical Engineering and Computer Science, Northwestern Univ., September 1989.

[11] C.-C. Chou and K. G. Shin. *A Distributed Table-Driven Route Selection Scheme for Establishing Real-Time Video Channels.* Submitted for publication.

[12] C.-C. Chou and K. G. Shin. *Multiplexing Statistical Real-Time Channels on a Multiaccess Network.* Submitted for publication.

[13] C.-C. Chou and K. G. Shin. *Statistical Real-Time Channels on Multiaccess Networks.* Submitted for publication.

[14] C.-C. Chou and K. G. Shin, "Statistical real-time video channels over a multiaccess network," *Proc. High-Speed Networking and Multimedia Computing Symposium, IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, February 1994.

[15] R. L. Cruz, *A Calculus for Network Delay and a Note on Topologies of Interconnection Networks*, PhD thesis, University of Illinois at Urbana-Champaign, July 1987.

[16] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Proc. SIGCOMM 89*, pp. 1–12, 1989.

[17] J. W. Dolter, *A Programmable Routing Controller Supporting Multi-Mode Routing and Switching in Distributed Real-Time Systems*, PhD thesis, Department of Elec. Eng. and Comput. Sci., The University of Michigan, Ann Arbor, MI 48109-2122, September 1993.

[18] 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, pp. 368–379, April 1990.

[19] D. L. Gall, "MPEG: A video compression standard for multimedia applications," *Communications of the ACM*, pp. 305–313, April 1991.

[20] R. V. Hogg and A. T. Craig, *Introduction to Mathematical Statistics*, Macmillan Publishing Co., Inc., 4th edition, 1978.

[21] V. C. Jones, *MAP/TOP Networking*, McGraw-Hill Book Company, 1987.

[22] D. D. Kandlur and K. G. Shin, "Design of a communication subsystem for HARTS," Technical Report CSE-TR-109-91, CSE Division, Department of Electrical Engineering and Computer Science, The University of Michigan, 1991.

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

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

[25] L. Kleinrock and H. Opderbeck, "Throughput in the ARPANET-protocols and measurement," *IEEE Trans. Communication*, vol. COM-25, no. 1, pp. 95–103, Jan 1977.

[26] J. F. Kurose, M. Schwartz, and Y. Yemini, "Multiple-access protocols and time-constrained communication," *ACM Computing Surveys*, vol. 16, pp. 43–70, 1984.

[27] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," *Proc. Real-time Systems Symposium*, pp. 166–171, 1989.

[28] A. Leon-Garcia, *Probability and Random Processes for Electrical Engineering*, Addison-Wesley Publishing Company, 1st edition, 1989.

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

[30] P. Pancha and M. E. Zarki, "A look at the MPEG video coding standard for variable bit rate video transmission," *INFOCOM*, 1992.

[31] A. K. J. Parekh, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*, PhD thesis, Department of Elec. Eng. and Comput. Sci., The Massachusetts Institute of Technology, February 1992.

[32] F. E. Ross, "FDDI–a tutorial," *IEEE Communications Magazine*, vol. 24, pp. 10–17, May 1986.

[33] 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.

[34] K. G. Shin and M.-S. Chen, "Performance analysis of distributed routing strategies free of ping-pong-type looping," *IEEE Trans. Comput.*, vol. C-36, no. 7, pp. 129–137, February 1987.

[35] K. G. Shin and M.-S. Chen, "Minimal order loop-free routing strategy," *IEEE Trans. Comput.*, vol. C-39, no. 5, pp. 870–888, July 1990.

[36] K. G. Shin, "Real-time communications in a computer-controlled workcell," *IEEE Transactions on Robotics and Automation*, pp. 105–113, February 1991.

[37] K. G. Shin and C.-C. Chou. *A Distributed Route-Selection Scheme for Establishing Real-Time Channels*. Submitted for publication.

[38] K. G. Shin and C.-C. Chou, "Design and evaluation of real-time communication for FieldBus based manufacturing systems," *1992 IEEE Local Computer Network Symposium*, pp. 483–492, September 1992.

[39] K. G. Shin and C.-C. Chou, "A simple distributed loop-free routing strategy for computer communication networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1308–1319, December 1993.

[40] J. K. Strosnider and T. E. Marchok, "Responsive, deterministic IEEE 802.5 token ring scheduling," *Journal of Real-Time Systems*, vol. 1, pp. 133–158, September 1989.

[41] A. S. Tanenbaum, *Computer Networks*, Prentice-Hall International, 2nd edition, 1989.

[42] M. Tangemann and K. Sauer, "Performance analysis of the timed token protocol of FDDI and FDDI-II," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 2, pp. 271–278, February 1991.

[43] A. Valenzano, C. Demartini, and L. Ciminiera, *MAP and TOP communications, standards and applications*, Addison-Wesley Publishing Company, 1992.

[44] D. C. Verma, *Guaranteed Performance Communication in High Speed Networks*, PhD thesis, University of California at Berkeley, 1991.

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

[46] J. Walrand, *Communication Networks: A First Course*, Irwin and Aksen Associates, 1991.

[47] H. Zhang and D. Ferrari, "Rate-controlled static-priority queueing," Technical Report TR-92-003, Computer Science Division, University of California at Berkeley, January 1992.

[48] Q. Zheng, K. G. Shin, and E. Abram-Profeta, "Transmission of compressed digital motion video over computer networks," *COMPCON Spring'93*, pp. 37–46, 1993.