

**PROTOCOLS FOR DISTRIBUTED
REAL-TIME SYSTEMS**

**Krishnamoorthy Arvind
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003**

**COINS Technical Report 91-42
May 1991**

**PROTOCOLS FOR DISTRIBUTED REAL-TIME
SYSTEMS**

A Dissertation Presented

by

KRISHNAMOORTHY ARVIND

**Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of**

DOCTOR OF PHILOSOPHY

May 1991

Department of Computer and Information Science

© Copyright by Krishnamoorthy Arvind 1991

All Rights Reserved

This research was supported in part by the Office of Naval Research under contract N00014-85-K-0398 and by the National Science Foundation under grants DCR-85003322 and CCR-8716858.

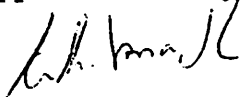
PROTOCOLS FOR DISTRIBUTED REAL-TIME SYSTEMS

A Dissertation Presented

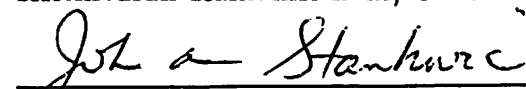
by

KRISHNAMOORTHY ARVIND

Approved as to style and content by:



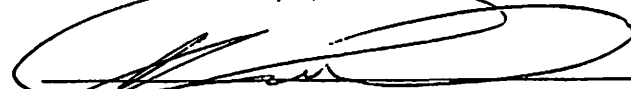
Krithivasan Ramamritham, Co-Chairman



John A. Stankovic, Co-Chairman



James F. Kurose, Member



Christos G. Cassandras, Member



W. Richards Adrion, Department Head
Department of Computer and Information Science

Dedicated to readers

ACKNOWLEDGEMENTS

I am deeply grateful to everyone who made this accomplishment possible. I would like to thank my advisors Professor Krithi Ramamritham and Professor Jack Stankovic for their support, interest, encouragement and guidance during the course of this research. It has been a great fortune for me to have had the opportunity to work with them. The readability of this thesis and the related research papers owes a great deal to their suggestions and comments. Their eagerness, patience, readiness and willingness to review various drafts of every piece of written material that I gave them was a source of great encouragement. I am also fortunate to have had as my other committee members, Professor Jim Kurose and Professor Christos Cassandras. They were always helpful and supportive. I am thankful to them for their helpful comments and suggestions. I consider myself lucky to have been a graduate student in the COINS department during Professor Rick Adrion's chairmanship. I will always remember his approachability, friendliness, consideration and supportiveness. I have had the opportunity to interact with various members of the COINS faculty during my graduate study here. I am grateful to them for their guidance, support and patience and for what they have taught me. Thanks to Renee for handling all the paper work related to graduate school and to Betty for mailing out all the letters and papers. Thanks to my family, fiancée, friends, colleagues in the CCS lab, and all my well wishers. Finally, my thanks to everyone else who deserves to be thanked, but whom I have missed above.

ABSTRACT

PROTOCOLS FOR DISTRIBUTED REAL-TIME SYSTEMS

MAY 1991

KRISHNAMOORTHY ARVIND

B.TECH., INDIAN INSTITUTE OF TECHNOLOGY

M.S., UNIVERSITY OF MASSACHUSETTS

Ph.D., UNIVERSITY OF MASSACHUSETTS

Directed by: Professor Krithivasan Ramamritham
Professor John A. Stankovic

Novel and specialized protocols will be necessary to deal with the requirements of time-constrained communication and synchronized clocks in an important area of technology, viz., the next generation of distributed real-time systems. Our research concentrates on developing distributed system *protocols* that realize these requirements. In this dissertation, we have proposed, analyzed and evaluated new protocols to meet these requirements.

A *clock synchronization* protocol is used to provide support for a common system-wide time base. We have proposed and analytically evaluated a novel *probabilistic algorithm* for clock synchronization, that can *guarantee a much lower bound* on the deviation between clocks than most existing algorithms. The guarantee offered by our algorithm is however probabilistic, i.e., there is a non-zero probability that the guarantee offered by our algorithm will fail to hold. The *probability of invalidity* of the guarantee, i.e., the probability that the deviation exceeds the guaranteed maximum deviation, can however be made *extremely small* by transmitting

a sufficient number of messages. We have presented a detailed analysis of the protocol. Among other things, we considered three different bounds on the probability of invalidity, and showed that a bound on the probability of invalidity decreases exponentially with the number of messages.

In our work on time-constrained communication, we have proposed RTLAN, a new local area network architecture for distributed real-time systems. RTLAN incorporates new communication abstractions (real-time virtual circuit, real-time datagram) and provides new classes of connection-oriented (RTCOS) and connectionless services (RTCLS) that explicitly consider the timing requirements of messages. It employs specialized *real-time communication protocols* at the medium access control layer to support these services. We have developed, analyzed and evaluated a homogeneous suite of five real-time medium access control protocols based on a uniform window splitting paradigm. Performance evaluation studies by simulation show that these protocols perform well compared to idealized baseline protocols.

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGEMENTS	v
ABSTRACT	vi
LIST OF TABLES	xiii
LIST OF FIGURES	xiv
 Chapter	
1. INTRODUCTION	1
1.1 Motivation and Goals	3
1.1.1 Clock Synchronization	3
1.1.2 Real-Time Communication	6
1.2 Research Contributions	11
1.2.1 Clock Synchronization	12
1.2.2 RTLAN	13
1.2.3 Real-Time MAC Protocols	13
1.3 Overview of the Dissertation	15
2. CLOCK SYNCHRONIZATION	17
2.1 Probabilistic Algorithms	17
2.1.1 Cristian's Algorithm	18
2.1.2 Our Algorithm	19
2.2 Description of Our Algorithm	20
2.2.1 Assumptions	20
2.2.2 The Algorithm	22
2.2.2.1 Transmitting Time	22

2.2.2.2	Clock Synchronization	23
2.2.2.3	Pseudo-Code	24
2.3	Summary of Analysis	27
2.3.1	Overview of Properties	28
2.3.2	Performance	30
2.3.3	Comparison	31
2.4	Summary	34
3.	ANALYSIS OF OUR CLOCK SYNCHRONIZATION PROTOCOL	37
3.1	Transmission Error	37
3.2	Probability of Invalidity	43
3.2.1	Experimental Bound	44
3.2.2	Tchebycheff Bound	44
3.2.3	Gaussian Approximation Bounds	45
3.3	Number of Messages	50
3.4	Bound on the Deviation between Clocks	52
3.5	Discussion	53
3.5.1	Independence	53
3.5.2	Moments of the End-to-End Delay	54
3.5.3	Performance and Omission Faults	56
3.5.4	Master Process Failures	56
3.6	Summary	57
4.	REAL-TIME COMMUNICATION	58
4.1	Introduction	58
4.2	Real-Time Communication	59
4.3	Local Area Network Architectures	62
4.4	RTLAN	66
4.5	RTLAN LLC Layer	67
4.5.1	Services	68

4.5.1.1	Real-Time Connection-Oriented Service	68
4.5.1.2	Real-Time Connectionless Service	70
4.5.2	Fragmentation	71
4.5.3	Guaranteeing Real-Time Connections	72
4.5.3.1	Priority Assignment Approach	72
4.5.3.2	Real-Time Virtual Circuit Approach	73
4.5.4	Flow Control and Error Control	76
4.6	RTLAN MAC Layer	78
4.6.1	RTCOS/MAC Protocols	79
4.6.1.1	Priority Resolution Protocols	79
4.6.1.2	Real-Time Virtual Circuit Protocols	82
4.6.2	RTCLS/MAC Protocols	85
4.7	Summary	87
5.	WINDOW MAC PROTOCOLS	89
5.1	Introduction	89
5.2	A Uniform Approach to MAC Protocols	93
5.2.1	Window Splitting Procedure	94
5.2.2	Pseudocode	98
5.2.3	Properties of window	99
5.3	The PRI Window Protocol	102
5.4	The RTDG Window Protocol	103
5.4.1	Pseudocode	103
5.4.2	Properties of RTDG	104
5.5	The RTVC Window Protocol	107
5.5.1	Pseudocode	108
5.5.2	Properties of RTVC	109
5.6	Integrated MAC Protocols	112

5.6.1	INTPVC	115
5.6.1.1	Pseudocode	116
5.6.1.2	Properties of INTPVC	117
5.6.2	INTPDG	117
5.6.2.1	Pseudocode	118
5.6.2.2	Properties of INTPDG	118
5.7	Discussion	119
5.7.1	Bandwidth Exchange	119
5.7.2	Ring Topology	120
5.8	Summary	122
6.	PERFORMANCE EVALUATION	127
6.1	Performance Metrics	127
6.2	Baseline Protocols	128
6.2.1	CML	129
6.2.2	INRT	129
6.2.3	IRTVC	130
6.2.4	VCTIMER	131
6.3	System Model	132
6.4	Simulation Results	133
6.4.1	RTDG Window Protocol	134
6.4.2	RTVC Window Protocol	136
6.4.3	Integrated Window Protocols	136
6.4.4	Simulation Results: A Comprehensive Presentation . . .	139
6.4.4.1	RTDG Window Protocol	141
6.4.4.2	RTVC Window Protocol	151
6.4.4.3	INTPVC Window Protocol	154
6.4.4.4	INTPDG Window Protocol	164
6.5	Summary	171

7. RESERVATION-BASED SCHEMES	177
7.1 Introduction	177
7.2 Reservation Protocols	178
7.2.1 Single Packet Reservation	178
7.2.2 Multiple Packet Reservations	179
7.2.3 Explicit Reservation Protocols	182
7.3 Guarantee-Based Reservation Protocols	187
7.4 Summary	190
8. CONCLUSION	191
8.1 Results and Contributions	191
8.1.1 Clock Synchronization	191
8.1.2 RTLAN	192
8.1.3 Real-Time MAC Protocols	193
8.2 Future Work	195
BIBLIOGRAPHY	198

LIST OF TABLES

Table		Page
2.1	$\text{erfc}(x)$	29
2.2	Number of synchronization messages versus p and $\frac{\epsilon_{\max}}{\sigma_d}$ ($n_g = 10$) .	32
4.1	Minimum real-time virtual circuit packet laxities (4 Mbps channel)	85

LIST OF FIGURES

Figure	Page
1.1 A mission control scenario: a distributed real-time system	6
2.1 Clock synchronization process at the master node	25
2.2 Clock synchronization process at a slave node	26
4.1 Cooperating team of robots	61
4.2 IEEE 802 LAN architecture	64
4.3 RTLAN architecture	66
4.4 Real-time connection	68
4.5 Real-Time connection requirements	69
4.6 Real-time virtual circuit scheduling	74
4.7 Abstractions provided by RTLAN layers	75
4.8 Packet service time	83
5.1 An example of the window procedure in operation	96
5.2 The window splitting procedure	97
5.3 The PRI window protocol	102
5.4 The RTDG window protocol	103
5.5 RTVC protocol state machine	106
5.6 The RTVC window protocol	108
5.7 Two queueing scenarios	113
5.8 INTPVC protocol state machine	116
5.9 The INTPVC window protocol	124
5.10 INTPDG protocol state machine	125
5.11 The INTPDG window protocol	126
6.1 Simulation study results: RTDG protocol	135
6.2 Simulation study results: RTVC protocol	137
6.3 Simulation study results: INTPVC and INTPDG	138
6.4 Simulation study results: effects of integration	140

6.5	Dependence of loss fraction on number of nodes ($\delta_{max} = 128$)	142
6.6	Dependence of loss fraction on number of nodes ($\delta_{max} = 1024$)	143
6.7	Dependence of loss fraction on number of nodes ($\delta_{max} = 16384$)	144
6.8	Dependence of loss fraction on window size ($N = 16$)	145
6.9	Dependence of loss fraction on window size ($N = 32$)	146
6.10	Dependence of loss fraction on window size ($N = 64$)	147
6.11	Dependence of waste fraction on window size ($N = 16$)	148
6.12	Dependence of waste fraction on window size ($N = 32$)	149
6.13	Dependence of waste fraction on window size ($N = 64$)	150
6.14	Comparison of RTDG, CML and INRT	152
6.15	Dependence of guarantee ratio on average laxity	153
6.16	Comparison of RTVC, IRTVC and TDMA ($N_{rtvc} = 16$)	155
6.17	Comparison of RTVC, IRTVC and TDMA ($N_{rtvc} = 32$)	156
6.18	Effect of integration on guarantee ratio ($\rho_{dg} = 0.50$)	157
6.19	Effect of integration on guarantee ratio ($\rho_{dg} = 1.00$)	158
6.20	Guarantee ratio: VCTIMER vs. INTPVC ($\rho_{dg} = 0.50$)	160
6.21	Guarantee ratio: VCTIMER vs. INTPVC ($\rho_{dg} = 1.00$)	161
6.22	Effect of integration on loss fraction ($\rho_{dg} = 0.50$)	162
6.23	Effect of integration on loss fraction ($\rho_{dg} = 1.00$)	163
6.24	Loss fraction: VCTIMER vs. INTPVC ($\rho_{dg} = 0.50$)	165
6.25	Loss fraction: VCTIMER vs. INTPVC ($\rho_{dg} = 1.00$)	166
6.26	Effect of integration on INTPDG guarantee ratio ($\rho_{dg} = 0.50$)	167
6.27	Effect of integration on INTPDG guarantee ratio ($\rho_{dg} = 1.00$)	168
6.28	Guarantee ratio: VCTIMER vs. INTPDG ($\rho_{dg} = 0.50$)	169
6.29	Guarantee ratio: VCTIMER vs. INTPDG ($\rho_{dg} = 1.00$)	170
6.30	Effect of integration on INTPDG loss fraction ($\rho_{dg} = 0.50$)	172
6.31	Effect of integration on INTPDG loss fraction ($\rho_{dg} = 1.00$)	173
6.32	Loss fraction: VCTIMER vs. INTPDG ($\rho_{dg} = 0.50$)	174
6.33	Loss fraction: VCTIMER vs. INTPDG ($\rho_{dg} = 1.00$)	175
7.1	Bit-mapped reservation protocol	179
7.2	Binder's broadcast satellite protocol	180

7.3 Reservation ALOHA 181
7.4 Roberts' reservation protocol 183
7.5 Reservation TDMA 184
7.6 Priority-oriented demand assignment 185

CHAPTER 1

INTRODUCTION

A real-time system is a system whose correctness depends not only on the logical results of computation, but also on the time at which the results are produced [92]. Activities in a real-time system have timing constraints associated with them which must be satisfied in order for the system to exhibit correct behavior. Real-time computing systems play an important role in society [88]. Examples of current real-time systems include nuclear power plants [2], industrial process control systems [67], air traffic control systems [25], robot controllers [72], the space shuttle avionics system [17], space mission control system ([71], [94]) and defense systems. The next generation of real-time systems, driven by advances in hardware and software technologies, will be even more sophisticated than today's systems. They will be complex, distributed and will be characterized by highly dynamic, adaptive and intelligent behavior [88]. This research addresses certain important requirements that arise in distributed real-time systems. In this introductory chapter, we develop the context for the research presented in this thesis, define the scope and objectives of our research and provide an overview of what we have accomplished. Later chapters fill in the details.

Distributed real-time systems based on local area networks are becoming increasingly prevalent in the research, military and commercial sectors. Distribution is an an important characteristic of the next generation of real-time systems [88]. This tendency towards distribution is driven by a number of factors including:

1. *Real-time requirements*: Distributed processing becomes necessary whenever the computational requirements and timing constraints of the tasks are such that, a single processor cannot meet the timing requirements of all the tasks.

2. *Fault-tolerance*: Fault-tolerance is an important requirement for real-time systems whose failure could result in catastrophic consequences. Fault-tolerance is typically achieved through the use of redundant components. A distributed system due to its inherent multiplicity of processing components is a natural choice for supporting fault-tolerance requirements.
3. *Performance-cost ratio*: Distributed processing offers the possibility of a tremendous improvement in performance at a low cost. For example, multimicroprocessor systems built using low cost off-the-shelf components can provide very high levels of performance and are becoming increasingly popular.
4. *Inherent parallelism in application domains*: Distributed processing is a natural choice for domains that offer scope for concurrency or in which physical distribution is inherent.

Activities in a *distributed* real-time system may be spread across physically separated nodes. This distribution gives rise to a number of system requirements of which two important ones, to which we restrict the scope of our research, are:

1. Support for a common system-wide time base.
2. Support for time-constrained communication between physically separated nodes.

Our research concentrates on developing distributed system *protocols* that realize these system requirements. Protocols are required to ensure the proper functioning of a system whenever the functioning involves the coordination of multiple concurrently executing agents. A protocol is a set of mutually agreed upon rules and conventions that the coordinating agents adhere to, and is typically expressed as a *distributed algorithm*¹. Examples of distributed system protocols abound in the literature and include clock synchronization protocols ([6], [7], [23], [26], [31], [47], [53], [54], [63]) communication protocols at the various network layers such as the Ethernet (CSMA/CD) Protocol [69], Internet Protocol (IP), Transmission Control

¹For this reason, we use the term *algorithm* and *protocol* interchangeably.

Protocol (TCP), various atomic broadcast protocols, and agreement protocols. In this research, we are specifically interested in protocols for clock synchronization and real-time communication. A *clock synchronization protocol* is used to provide support for a common system-wide time base. Real-time *communication protocols* are used to provide support for time-constrained communication.

1.1 Motivation and Goals

The overall aim of our research is to develop protocols that support the requirements of a global time base and of time-constrained communication in a distributed real-time system. In this section, we identify the motivation and goals of our research. We consider the problem of clock synchronization in Section 1.1.1 and of real-time communication in Section 1.1.2.

1.1.1 Clock Synchronization

A fault-free hardware clock (HC), even if initially synchronized with a standard time reference, tends to drift away from the standard over a period of time. If such a clock is used to measure a time interval (t_1, t_2) , the measured length of the interval $(HC(t_2) - HC(t_1))$ tends to differ from the actual length of the interval. However, the magnitude of the fractional error in the measurement of the interval is bounded by a constant ρ_a called the *maximum drift rate* of the clock:

$$\left| \frac{(HC(t_2) - HC(t_1)) - (t_2 - t_1)}{(t_2 - t_1)} \right| < \rho_a. \quad (1.1)$$

ρ_a is typically of the order of $1 \mu\text{sec}/\text{sec}$; thus an initially synchronized fault-free clock could drift away from the reference by a few seconds every few days, in the worst case. A *clock synchronization protocol* is used in a distributed system, to synchronize clocks that can drift apart, to ensure that the maximum deviation between the clocks is bounded.

Clock synchronization has been an active area of research in distributed systems ([6], [7], [23], [26], [28], [31], [47], [48], [52], [53], [54], [63], [64], [101], [105]).

It is of special importance in distributed real-time systems, where an event may spread across multiple nodes each with its own local clock. In order to specify the timing constraints of such events, a common reference of time is necessary in a distributed real-time system [47]. A clock synchronization protocol is used to generate such a common reference of time. Synchronized clocks also have other applications such as determining a total order on the events in a distributed system [52] and in performance measurement in distributed systems [26].

Several protocols have been proposed for clock synchronization ([23], [26], [31], [47], [53], [54], [63]). However all these protocols with the exception of the one in [23] are limited in the upper bound (γ_{max}) on the deviation between clocks in the system that they can guarantee, by certain bounds derived in [28] and [64]. Lundelius and Lynch [64] have shown that, even with no failures and with clocks that do not drift, for a system of N clocks (that are not initially synchronized), the γ_{max} that can be guaranteed has a lower bound given by:

$$\gamma_{max} \geq (d_{max} - d_{min}) \left(1 - \frac{1}{N}\right) \quad (1.2)$$

(Dolev, Halpern and Strong [28] have proved a related result for the closeness of synchronization obtainable along the real time axis). This bound implies that no clock synchronization algorithm can guarantee with *certainty*, a skew between clocks that is smaller than the right hand side of Eq. (1.2). The quantities d_{max} and d_{min} , in Eq. (1.2), refer to the maximum and minimum possible values respectively, of the end-to-end delay (d) of synchronization messages. The end-to-end delay of a message, which is the sum of the times required for a message to be prepared, sent to a receiver and be processed by the receiver, is a random delay determined by the amount of communication and computation going on in the system at the time the message is being sent, and by random events such as transmission errors that necessitate the retransmission of a message, context switches and page faults.

The value of this lower bound in a typical system based on a local area network is of the order of 50 millisecc [23]. This means that the smallest maximum skew that can be guaranteed with certainty and hence the resolution of the global time

base are also of the same magnitude. It is possible to improve the granularity of the resolution with special hardware support ([47]). However we are interested in investigating the possibility of achieving improved resolution using alternative approaches that *do not employ special hardware*. Thus a goal of our research effort is to:

Investigate new approaches to clock synchronization that do not use special hardware and that are yet capable of guaranteeing much smaller clock skews than algorithms that are constrained to obey Eq. (1.2).

The direction for our research on clock synchronization is derived from the observation that the bounds derived in [28] and [64], apply only to *deterministic* clock synchronization algorithms, i.e., algorithms that guarantee a γ_{max} with *certainty*. If we relax the requirement of certainty and permit an algorithm to provide a probabilistic guarantee (i.e., the guarantee provided by the algorithm may fail to hold sometimes, but with a *failure probability* that is known or that has a known bound), then we can find algorithms that can guarantee a γ_{max} which need not conform to Eq. (1.2). However, for a probabilistic guarantee to be useful, it must be possible to reduce the failure probability to any desired level by choosing the parameters of the algorithm suitably. Clock synchronization algorithms that can provide such a probabilistic guarantee on γ_{max} are referred to as *probabilistic*² clock synchronization algorithms. We have taken a probabilistic approach in our research on clock synchronization.

²The word 'probabilistic' refers to the uncertainty in the guarantee offered by the algorithm and is different from the sense in which the word is used in [28], where a probabilistic algorithm is an algorithm whose behavior itself is randomly determined.

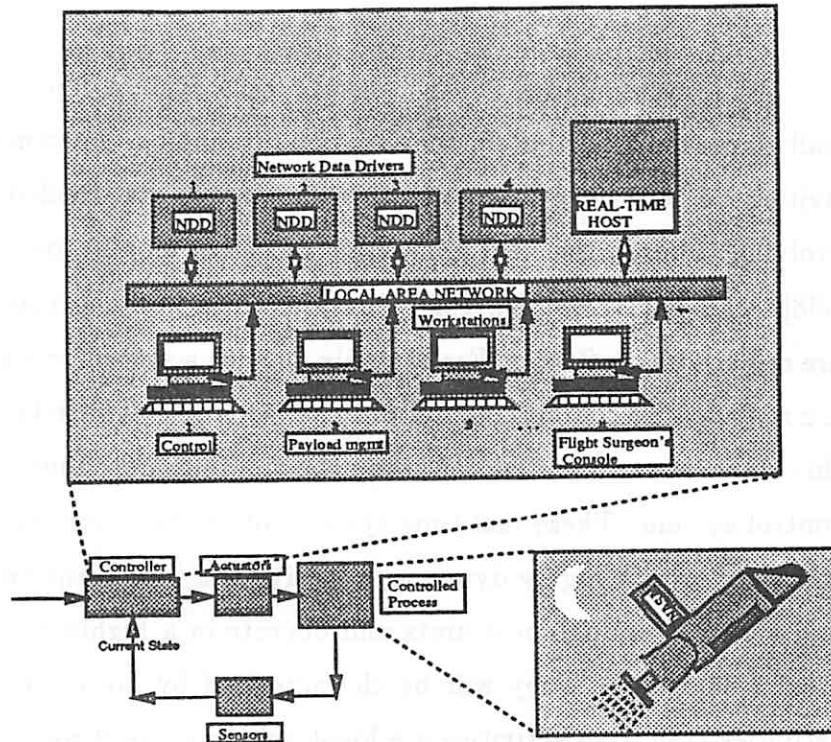


Figure 1.1 A mission control scenario: a distributed real-time system

1.1.2 Real-Time Communication

Distributed real-time systems based on local area networks³ are becoming increasingly prevalent in the research, military and commercial sectors. The space shuttle mission control system depicted in Figure 1.1 (adapted from [94]) is an example of such a system. This system consists of a number of computing nodes interconnected by a local area network. In this system, network data drivers transmit raw telemetry data received from space to a real-time host computer for processing. Mission controllers responsible for various aspects of the mission, with the help of the host, work stations, and the processed telemetry data, monitor and control the operation of the mission.

³Local area networks are typically used with distributed real-time systems because of the limited geographical span of the systems. However, even applications with a wider geographical span may be based on local area networks. For example, in air traffic control the entire air space is divided into smaller units (sectors) and the traffic within each unit is controlled by a distributed real-time system based on a local area network [25].

While systems such as the mission control system depicted in Figure 1.1 are typically large and distributed, they are not autonomous since many of the high-level activities are performed by cooperating human experts. Real-time systems are steadily evolving towards the next generation of *closed loop* autonomous real-time systems ([38], [11], [55], [71], [82], [102], [104]) in which human experts in the feedback loop are replaced by software. For example, in such a system, cooperating human experts are replaced by communicating problem-solving software tasks. The culmination of this trend towards autonomy would be the fully autonomous closed loop real-time control system. These real-time systems of the next generation will be distributed, complex, exhibit highly dynamic, adaptive and intelligent behavior and possess several types of timing constraints and operate in a highly non-deterministic environment ([88], [90]). They will be characterized by both *reflexive* and *cognitive* [56] activities (reflexive activities are low-level life-supporting activities such as sensory processing that are typically periodic and static; cognitive activities are high-level dynamic activities, e.g., cooperative problem solving and planning).

Many of the activities in these systems will have hard timing constraints associated with them. A failure in meeting the timing constraints of these activities may result in catastrophic consequences [88]. Consequently, *predictability*, the ability to determine⁴ *a priori* whether or not the timing requirements of an activity can be met is an important requirement in these systems. Communication between tasks resident on the same or different nodes is an important activity in such systems. Messages characterized by timing constraints will be exchanged between both application tasks and operating system tasks. Consequently, the operating system and the underlying network in a distributed real-time system must provide various facilities that support predictability and that take into account timing constraints of messages. Our research addresses some of the communication requirements that arise in such systems.

Communication in a distributed *real-time* system differs from *non-real-time* systems because of the introduction of explicit timing constraints. Individual messages

⁴Prior to system startup for statically specifiable activities and prior to activity commencement for dynamic activities.

have timing constraints associated with them and different kinds of requirements for the satisfaction of these constraints. Two classes⁵ of time-constrained messages arise in such systems:

1. *Guarantee Seeking Messages*: These are messages typically critical or essential for the proper operation of the system. The requirements of these messages include a *guarantee* from the system that, if the activity that gives rise to them is accepted for execution, their timing constraints will be met with certainty. This class of messages requires mechanisms that provide predictability, i.e., that make it possible to *a priori* verify that or determine if the timing constraints of such messages will be met.
2. *Best Effort Messages*: These are messages, typically with soft timing constraints, that do not require a *guarantee* from the system that their timing constraints will be met. However the system will try its best to satisfy the timing constraints of these messages, since minimizing the number of such messages whose timing constraints are violated will result in increased value (in some sense) for the system.

In order to support these two classes of messages, the operating system and the communication network underlying the distributed real-time system must provide various kinds of communication services. One of the goals of this research is to:

Identify useful communication services and abstractions in a distributed real-time environment.

As part of this research, we have developed new classes of local area network services known as *real-time connectionless service* (RTCLS) and *real-time connection-oriented service* (RTCOS) and the abstractions of *real-time virtual circuit* and

⁵A related taxonomy is described in [65] where time-constrained messages are classified into *critical*, *essential* and *nonessential* messages on the basis of the impact on the system performance of missing the deadline of a message. The first class of messages in our classification encompasses the critical and essential messages in [65], while the second class corresponds to the nonessential messages in [65].

real-time datagram (terms introduced in [92]) RTCOS is suitable for the class of guarantee-seeking messages while RTCLS is suitable for the class of best effort messages. Implementation of RTCOS and RTCLS requires support from the medium access control layer, in the form of specialized real-time medium access control protocols. Implementation of RTCOS requires support in the form of priority resolution protocols that implement global packet-level priority resolution, and protocols that can *guarantee bounded channel access times*. Implementation of RTCLS requires medium access control protocols that *explicitly consider the timing requirements of messages* such as message deadlines, and arbitrate access to the channel on a best efforts basis on the basis of these requirements. Thus one of the goals of this research effort is to

Develop medium access control protocols that support time-constrained communication in a local area network.

In our research, we have mainly focussed on developing medium access control protocols for a local area network based on a bidirectional bus topology, a popular topology for real-time systems. However we do briefly discuss the implementation of the proposed services and abstractions on a ring topology.

The starting point for our work was provided by the MLF window protocol in [108]. Zhao, Stankovic and Ramamritham [108] have suggested the use of a minimum laxity first policy for channel arbitration in a multiple access network to minimize the number of messages that miss their deadlines. Marinescu and Szpankowski [66] have also suggested the use of a minimum laxity first policy for channel arbitration in a multiple access network, based on results from [73]. In [73], the minimum laxity first service discipline (also known as the shortest time to extinction policy) or its variations have been shown to maximize the fraction of the number of customers that meet their deadlines in a queueing system under fairly general conditions. Thus the minimum laxity first policy for channel arbitration is a good choice for implementing the best effort real-time connectionless services. A window protocol can be used to closely approximate the system-wide minimum laxity first policy for message transmission over a shared bus [108]. Thus a window protocol is well-suited

for providing real-time connectionless services. However the window protocol used to implement the global minimum laxity first policy cannot guarantee bounded channel access times for nodes, in general. Thus this protocol, as it is, is not suitable for implementing real-time connection-oriented services. An approach that makes use of the window splitting paradigm to implement real-time connection-oriented services also is attractive for the following reason. Such an approach makes it possible to use the same medium access control logic and LAN controller hardware that is used to support real-time connectionless services, to support real-time connection-oriented services also. This advantage is especially important when both classes of services are to be supported by a single protocol in an integrated manner. This motivates us to

Explore generalizations and adaptations of the real-time window protocol that can be used to support RTCOS also.

The PRI, RTVC and RTDG window protocols described in this thesis are products of our effort in this direction. The PRI window protocol and the RTVC window protocol can be used to support RTCOS, while the RTDG window protocol can be used to support RTCLS.

The PRI, RTVC and RTDG protocols are suitable for supporting either RTCOS or RTCLS traffic, but not both. In this research, we also examine integrated protocols that may be used to support both classes of traffic in a unified manner. Kurose, Schwartz and Yemini [50] point out the importance of integrated protocols in multimedia applications. Integrated handling of multiple classes of traffic has a number of advantages, related to both physical implementation and performance. In terms of physical implementation, an integrated approach permits the use of a common interface and common cabling for all the classes of traffic. In addition to cost benefits, this also results in conservation of chassis slots since the number of network interface boards is reduced. Use of an integrated protocol also offers a performance advantage by making available the bandwidth unused by one class of traffic to the others.

The potential advantages that integrated protocols have to offer motivate the following research goal:

Develop new integrated medium access control protocols that can be used to support both real-time connection-oriented service traffic and real-time connectionless service traffic on a common bus.

The integrated window protocols INTPVC and INTPDG developed in this thesis are products of our effort in this direction.

Another goal of our research on real-time communication is to

Evaluate the performance of the protocols developed.

Our approach to performance evaluation is based on event-driven discrete-event simulation. The primary performance indices that we use to measure the efficacy of our protocols are the *guarantee ratio* and the *loss fraction* ([107], [108], [65]). The *guarantee ratio* is defined as the fraction of the number of packets that arrive whose deadlines are guaranteed to be met. This performance index is used to measure the quality of real-time connection-oriented services. The *loss fraction* is defined as the ratio of the number of packets that miss their deadlines to the sum of the number of packets that miss their deadlines and the number that are successfully transmitted. This quantity is used to measure the performance of real-time connectionless services.

1.2 Research Contributions

This research deals with two important problems that arise in distributed real-time systems and has generated novel solutions to the problems. The contributions of this research include a new protocol for clock synchronization based on a probabilistic approach, identification of several new real-time communication services (RTCOS and RTCLS) and abstractions (real-time virtual circuit and real-time

datagram), a new local area network architecture (RTLAN) for distributed real-time systems and a number of new real-time medium access control protocols (PRI, RTDG, RTVC, INTPVC and INTPDG) for supporting the new classes of services and abstractions. An integral part of the research is the validation of the proposed protocols through analysis and extensive simulation studies. Below, we briefly summarize the contributions and results of this research.

1.2.1 Clock Synchronization

We have developed a *new protocol for clock synchronization* in distributed real-time systems that can achieve much smaller clock skews than most existing protocols. This clock synchronization protocol performs much better than deterministic protocols proposed in the literature. We demonstrate, in our examples, that our algorithm is capable of guaranteeing a maximum deviation between clocks of the order of a few milliseconds, compared to the smallest maximum deviation of 50 milliseconds that can be guaranteed by deterministic algorithms. However the guarantee offered by our algorithm is probabilistic, i.e., there is a non-zero probability that the guarantee offered by our algorithm will fail to hold. The probability of invalidity, however, can be made *extremely* small by transmitting a sufficient number of messages. In our examples, we are able to achieve a probability of invalidity of (1×10^{-9}) by transmitting 10 synchronization messages.

We have presented a detailed analysis of the proposed clock synchronization protocol. Among other things, we have derived an expression for the minimum number of synchronization messages required to guarantee a specified maximum deviation at resynchronization and a specified probability of invalidity. We have considered three analytical bounds on the probability of invalidity, viz., the *Tchebysheff*, *error function* and *exponential* bounds, and showed that the probability of invalidity decreases exponentially (or better) with the number of synchronization messages.

1.2.2 RTLAN

Most current work in real-time communication deals with static systems in which messages with timing constraints are mainly periodic, or occur only rarely. However the dynamic-closed loop distributed real-time systems of the future will be characterized by richer communication patterns. Specialized network services and protocols will be required to support the communication requirements of these systems. In this dissertation, we have proposed RTLAN, *a new local area network architecture* for communication in such systems. RTLAN is targeted for complex real-time applications which have time-constrained communication requirements that range from simple best effort delivery requirements to dynamic guarantees of general timing requirements. Some of the distinguishing features of RTLAN include the provision of *new classes of connection-oriented and connectionless* services known as real-time connection-oriented service (RTCOS) and real-time connectionless service (RTCLS) respectively, that take the timing constraints of messages explicitly into account, incorporation of the notions of *a priori* scheduling and guarantee in the logical link control layer, and the use of specialized real-time protocols at the medium access control layer. We have developed the useful *abstractions* of real-time virtual circuits and real-time datagrams (terms introduced in [92]) in the context of a local area network.

1.2.3 Real-Time MAC Protocols

We have considered a distributed real-time system based on a local area network with a *bus* topology, and proposed a *new homogeneous suite of real-time medium access control protocols* for supporting RTCOS and RTCLS, that is devoid of deficiencies found in existing standards for medium access control protocols for bus-based systems, viz., the IEEE 802.3 standard and the IEEE 802.4 standard. (the IEEE 802.3 and 802.4 standards do not support individual packet level priority resolution or real-time datagram arbitration; the IEEE 802.3 standard does not guarantee bounded channel access times also). The proposed suite consists of five protocols, viz., PRI, RTDG, RTVC, INTPVC and INTPDG, all based on a uniform window

splitting paradigm. The protocol PRI implements priority-based arbitration. The protocol RTDG implements real-time datagram arbitration based on the minimum laxity first policy. The protocol RTVC implements real-time virtual circuit arbitration. The protocols INTPVC and INTPDG are integrated protocols that may be used to support both real-time datagram and real-time virtual circuit arbitration in an integrated manner on a common bus. We have analyzed the window protocols and derived various properties exhibited by the protocols. We have also conducted a simulation study to evaluate the performance of the protocols (The model of the system assumed and the various assumptions made in this study are detailed in Chapter 6). The main properties of the protocols, derived from the analytical and simulation study are summarized below:

1. The per packet contention resolution overhead for the window-based contention resolution procedure used by the window protocols increases logarithmically with the size of the window.
2. The PRI window protocol not only provides a capability, viz., system-wide packet level priority-based arbitration, that the 802.3 and 802.4 standard protocols lack, but also is characterized by a slightly smaller average priority resolution overhead than other non-standard protocols that have been proposed for priority resolution.
3. The RTDG protocol closely approximates the minimum laxity first policy for channel arbitration.
4. The performance of the RTDG window protocol is superior to that of an idealized baseline protocol that provides a bound on the performance achievable by any non-real-time protocol (i.e., a protocol that has no notion of packet timing constraints).
5. The performance of the RTVC window protocol is close to that of an idealized baseline protocol that provides a bound on the performance achievable by any protocol that can guarantee bounded channel access times.

6. The integrated window protocol INTPVC is characterized by a smaller (by a factor of about 2) worst case channel access time for real-time virtual circuit packets, than the standard 802.4 token bus protocol operating in integrated mode (priority mode) or the FDDI protocol.
7. The quality of service provided to real-time virtual circuit packets by the integrated window protocol INTPVC is better than that provided by VCTIMER, an idealized baseline integrated protocol.
8. The integrated window protocol INTPDG is characterized by a worst case channel access time for real-time virtual circuit packets, that is about the same as (but larger by about two packet transmission times) that for the 802.4 token bus protocol operating in priority mode or the FDDI protocol.
9. Use of an integrated protocol results in an improvement in the quality of performance. INTPVC provides a better quality of performance for real-time virtual circuit packets than RTVC. INTPDG provides a better quality of performance for real-time datagram packets than RTDG.

In addition to overcoming deficiencies of existing standards, and to the good performance characteristics pointed out above, the protocol suite proposed in this dissertation also has the advantage of structural homogeneity (thereby enabling the use of a uniform medium access control logic and LAN controller hardware), since all the protocols are based on a uniform window splitting paradigm.

1.3 Overview of the Dissertation

We have identified the goals of our research in this chapter. Our goals center around two important problems that arise in distributed real-time systems, viz., clock synchronization and communication. Our work towards realizing these goals has resulted in the development of a novel probabilistic clock synchronization protocol that can guarantee much smaller clock skews than most existing protocols, a new local area network architecture for communication in distributed real-time systems,

and new protocols for real-time communication. We develop the probabilistic clock synchronization protocol in Chapter 2. In Chapter 3, we provide a detailed analysis of the protocol and derive various properties of the protocol, including an expression for the minimum number of synchronization messages required to guarantee a specified maximum deviation between clocks, and various bounds on the probability of invalidity. In Chapter 4, we consider the problem of real-time communication and develop RTLAN, a local area network architecture for communication in distributed real-time systems. The notions of real-time connection-oriented service, real-time connectionless service, real-time virtual circuits and real-time datagrams are developed in this chapter. In Chapter 5, we consider the medium access control layer of RTLAN and develop a new homogeneous suite of five real-time medium access control protocols, viz., PRI, RTDG, RTVC, INTPDG and INTPVC, all based on a uniform window splitting paradigm. Chapter 6 addresses the performance evaluation aspects of these protocols. In Chapter 7, we consider the pessimism inherent in implementing real-time connection-oriented service using the notion of real-time virtual circuits and examine an alternative approach based on reservation. We conclude the dissertation in Chapter 8 with a summary of the results and contributions of this research, and an identification of directions for future research.

CHAPTER 2

CLOCK SYNCHRONIZATION

We introduced the need for clock synchronization in Chapter 1. A clock synchronization protocol is used in a distributed real-time system, to synchronize clocks that can drift apart, to ensure that the maximum deviation between the clocks is bounded. We also discussed the bound (Eq. 1.2) established by Lundelius and Lynch, viz., even with no failures and with clocks that do not drift, for a system of N clocks, the upper bound (γ_{max}) on the deviation between clocks that can be guaranteed has a lower bound given by:

$$\gamma_{max} \geq (d_{max} - d_{min}) \left(1 - \frac{1}{N}\right),$$

where d_{max} and d_{min} refer to the maximum and minimum possible values respectively, of the end-to-end delay (d) of synchronization messages.

2.1 Probabilistic Algorithms

The bound derived by Lundelius and Lynch, however apply only to *deterministic* clock synchronization algorithms ([26], [31], [47], [53], [54], [63]), i.e., algorithms that guarantee a γ_{max} with *certainty*. If we relax the requirement of certainty and permit an algorithm to provide a probabilistic guarantee (i.e., the guarantee provided by the algorithm may fail to hold sometimes, but with a *failure probability* that is known or that has a known bound), then we can find algorithms that can guarantee a γ_{max} which need not conform to Eq. (1.2). However for a probabilistic guarantee to be useful, it must be possible to reduce the failure probability to any desired level by choosing the parameters of the algorithm suitably. Clock synchronization algorithms that can provide such a probabilistic guarantee on γ_{max} are referred to as *probabilistic* clock synchronization algorithms.

2.1.1 Cristian's Algorithm

Cristian [23] has proposed one (probably the first) such probabilistic clock synchronization algorithm that is capable of guaranteeing much lower γ_{max} 's than deterministic algorithms.

Cristian's algorithm belongs to the class of non-averaging¹ clock synchronization algorithms and is based on a new remote clock reading method. This clock reading method can be used by nodes in a distributed system to read the clock at any other node in the system with an *a priori* specified maximum error, in spite of unpredictable message delays. The remote clock reading method consists of sending a time query to the remote node, measuring the elapsed time (known as the *round trip delay*) between the instant that the query is sent and the instant that a reply is received from the remote node, and estimating the reading on the remote clock on the basis of the received reply and the measured round trip delay. This method guarantees [23] that the maximum reading error is approximately $D - d_{min}$, where D is half the measured round trip delay. If the specified maximum error is $U - d_{min}$, then the node that is attempting to read the remote clock repeats the clock reading method at periodic intervals (of length W) until $D \leq U$. The node that is trying to read the remote clock is said to have reached *rapport* with the remote node when $D \leq U$. Note that there is a non-zero probability of failing to reach *rapport* if there is a restriction on the maximum number of clock reading attempts. However this probability can be made very small by increasing the maximum number of attempts permitted.

The clock reading method described above is used to implement a master-slave clock synchronization algorithm that guarantees a specified maximum deviation between clocks in the system. One node in the system is designated as the master node and the rest of the nodes are designated as slave nodes. Each slave node

¹Clock synchronization algorithms can be classified into averaging or non-averaging algorithms on the basis of whether or not they involve some kind of averaging of information received from the various nodes. The algorithms proposed in ([47],[54],[63]) are examples of averaging algorithms while the algorithms proposed in ([23],[26],[31]) are examples of non-averaging algorithms.

periodically resynchronizes its clock with that of the master node by reading the master's clock using the clock reading method described above and adjusting its clock accordingly. This scheme guarantees a maximum deviation between the clocks at the master node and the slave node of approximately $U - d_{min} + \rho kW$, where k is the maximum number of attempts to reach rapport permitted and W is the length of the interval between successive attempts to reach rapport. This guarantee is probabilistic however, since there is a non-zero probability of not reaching rapport, i.e., of failing to resynchronize.

2.1.2 Our Algorithm

In ([6], [7]), we propose an algorithm that represents a new approach to probabilistic clock synchronization, based on averaging end-to-end delays of synchronization messages. Our algorithm makes use of a new probabilistic time transmission protocol, that may be used to transmit the time on the clock at one node in a distributed system to another node in the system with an *a priori* specified accuracy, in spite of random message delays. The protocol involves the transmission of multiple synchronization messages and works by smoothing out the random variations in the end-to-end delays of the synchronization messages through a process of averaging. This time transmission protocol is then incorporated into a master-slave scheme to implement a clock synchronization algorithm. One node in the system designated as the master, gets the clocks at the other nodes to synchronize with its own clock by periodically executing the time transmission protocol. Our algorithm, like Cristian's algorithm, can guarantee a much lower γ_{max} than the deterministic algorithms that have to conform to Eq. (1.2). Also, the probability that the guarantee provided will fail to hold good can be reduced (with an *exponential* rate of decrease) to a very small value by increasing the number of synchronization messages.

The rest of this chapter is organized as follows. In the next section, we describe our algorithm. We first state the assumptions underlying our work. We then discuss the proposed time transmission protocol (TTP) and describe how a clock synchronization algorithm can be constructed from it. In Section 2.3, we provide an

overview of the main properties of the algorithm and illustrate its performance with numerical examples. We conclude the chapter in Section 2.4 with a brief summary.

2.2 Description of Our Algorithm

2.2.1 Assumptions

We make the following assumptions about clocks, processes and message delays:

1. The system consists of N hardware clocks HC_i , with one clock synchronization process associated with each clock. The synchronization processes have read-only access to their own hardware clocks.
2. The hardware clocks have a much higher resolution than the delays involved (e.g., average end-to-end delay of messages, standard deviation of the end-to-end delays) and the deviation between clocks desired. For example, if these quantities are of the order of milliseconds, then the hardware clocks have a microsecond resolution.
3. The hardware clocks are ρ -bounded, i.e., they obey Eq. (1.1), which can be rewritten as follows:

$$(1 - \rho_a)(t_2 - t_1) \leq HC(t_2) - HC(t_1) \leq (1 + \rho_a)(t_2 - t_1) \quad (2.1)$$

where ρ_a is the *maximum* drift rate of the clocks. (ρ -boundedness is normally guaranteed by the manufacturer). It follows that the amount by which two ρ -bounded clocks can drift apart from each other during an interval of time τ is bounded by $(1 + \rho_a)\tau - (1 - \rho_a)\tau = 2\rho_a\tau$. We refer to the term $2\rho_a$ as the *maximum relative clock drift rate* (ρ).

4. The *actual* drift rate of a clock at any moment ($\hat{\rho}_a$), defined as the fractional error in the measurement of a small interval of time may possibly vary with time (due to change in physical conditions, etc.). Of course, $\hat{\rho}_a < \rho_a$. Analogous to the maximum relative drift rate ρ , we define an *actual* relative drift

rate $\hat{\rho} = 2\hat{\rho}_a$. We do not require that the actual drift rate $\hat{\rho}_a$ be known. However, in order to simplify the analysis, we assume that the variation in the actual drift rate over a transmission period (defined in Section 2.2.2.1) is negligible. The transmission period is typically in the range of a fraction of a minute.

5. Each synchronization process maintains one or more (depending on whether continuous or instantaneous adjustment is used) *logical* clocks, each of which is a linear function of the time on the hardware clock of the process. In the rest of the chapter, unless otherwise stated, the term 'clock' always refers to the logical clock (or the current logical clock if instantaneous adjustment is used) of the process under consideration.
6. Processes and communication links are free from faults. (We make this assumption initially, but in Section 3.5, we sketch schemes to handle performance, omission and crash failures. We assume in this chapter that, the probability of observing other types of faults (such as Byzantine faults) during the lifetime of the system can be made very small, by building sufficient error-detecting, error-correcting redundancy into the components of the system [26]).
7. The end-to-end message delay (d) between two processes can be modeled as a random variable. We do not require that the delay conform to any specific distribution. We assume that estimates \bar{d} and σ_d , of the expected value and standard deviation respectively of the end-to-end delay, are known. We also assume, in our analysis, that the end-to-end delays of the synchronization messages involved in our algorithm are independent of each other. These assumptions are discussed further in Section 3.5.

2.2.2 The Algorithm

We first describe how a process M can transmit the time on its clock to a process S with a maximum *transmission error* of ϵ_{max} . We then describe how this method of transmitting time can be used to construct a clock synchronization algorithm.

2.2.2.1 Transmitting Time

A process M can transmit the time on its clock to a process S by sending n synchronization messages according to the time transmission protocol, TTP, described below. The length of the time period starting at the time the first message is sent and ending at the time the n th message is sent is referred to as the *transmission period* (τ). On receiving the n th message, S makes an estimate T_{est} of the time on M's clock. The *transmission error* ϵ is defined as the difference between the estimate, that S makes of the time on M's clock, and the actual time T_{act} on M's clock at the time S makes the estimate, i.e.,

$$\epsilon = T_{est} - T_{act} \quad (2.2)$$

TTP works as follows. When process M wants to transmit the time on its clock to process S, it starts sending synchronization messages, msg_i , to S. The i th message, msg_i , which is sent at time T_i on its clock, is of the form "Time is T_i ". M sends n such messages within the transmission period τ , with the transmissions of the i th and $i + 1$ th messages being separated by an interval of time $W_{i,i+1}$. At the S end, each time a synchronization message, msg_i , is received, S records its time of receipt, R_i , according to its local clock. After S has received n messages, it computes the averages of the times on the messages and the receipt times as

$$\bar{T}(n) = \frac{1}{n} \sum_{i=1}^n T_i$$

and

$$\bar{R}(n) = \frac{1}{n} \sum_{i=1}^n R_i.$$

It then estimates the time on M's clock as:

$$T_{est} = R_n - \bar{R}(n) + \bar{T}(n) + \bar{d} \quad (2.3)$$

2.2.2.2 Clock Synchronization

The time transmission protocol that we described in the previous section can be used to construct a clock synchronization algorithm, in a manner similar to how the remote clock reading method is converted to a master-slave clock synchronization algorithm in [23]. As in [23], one process in the system is designated as the *master*, and the clocks of the other processes (the *slaves*) are synchronized to the clock of this process using TTP. However, unlike the algorithm in [23] where each slave process is individually responsible for getting its clock to synchronize with the clock of the master process, in our algorithm, the master process is responsible for getting the clocks of each of the slave processes to synchronize with its clock.

The master process gets the clock of a slave process to synchronize with its own clock as follows. It transmits the time on its clock to the slave process using the time transmission protocol TTP. The slave, when it has received n messages, estimates the time on the master's clock as in TTP and computes the adjustment $T_{est} - R_n$ that it needs to make to its clock in order to synchronize it with the master's clock, and then makes the adjustment. The slave can make the adjustment to its clock either *instantaneously* at this point (this point in time is referred to as the *resynchronization point*) or *continuously* over a period of time. The latter of these two alternatives is the preferred one, since instantaneous adjustment has disadvantages like the possibility of a backward correction of the time and the need to maintain multiple logical clocks [47].

If the adjustment is made instantaneously, the clock of the slave process reads T_{est} at the resynchronization point (immediately after resynchronization). The deviation between the clocks of the slave process and the master process at the resynchronization point is then equal to $T_{est} - T_{act}$ which is the same as the transmission error. Thus the transmission error ϵ represents the deviation between the clocks of

the slave process and the master process at the resynchronization point (immediately after resynchronization), if instantaneous adjustment is used. Hence we also refer to the transmission error ϵ as the *deviation at resynchronization*. In order to compensate for drift between the clocks of the master and slave processes that develops after resynchronization, the master process repeats this synchronization algorithm at periodic intervals of length R_{synch} (the *resynchronization interval*).

The master process executes the above algorithm for each slave process in the system. The master process, by ensuring that the clocks of all the slave processes are synchronized to its own clock, ensures that all the clocks in the system are mutually synchronized. The maximum deviation between the clocks of any two processes in the system is less than or equal to twice the maximum deviation between the master clock and some slave clock. Note that if the system is based on a *broadcast* network such as Ethernet, then the master process need transmit only one set of synchronization messages per resynchronization interval. Thus the number of synchronization messages required in such a system will be independent of the number of nodes in the system.

We show in Chapter 3 that our algorithm ensures that the average deviation at resynchronization can be made negligible compared to the maximum deviation at resynchronization. We also show that the maximum deviation at resynchronization (ϵ_{max}) and the probability (p) that the actual deviation at resynchronization exceeds this value (known as the *probability of invalidity*) can be made very small by increasing the number (n) of synchronization messages transmitted.

2.2.2.3 Pseudo-Code

A concise description of our algorithm may be found in Figures 2.1 and 2.2. Figure 2.1 contains the pseudo-code for the clock synchronization process at the master node M and Figure 2.2 the pseudo-code for the clock synchronization process at a slave node S. In what follows, we use the notation (M.j) to refer to line j of Figure 2.1 and (S.j) to refer to line j of Figure 2.2.

```

1. process Master(n:integer,  $R_{synch}$ :time)
2.   var i: integer;
3.      $\Theta$ : time;
4.     resynch, transmit, which-timer: timer;
5.   begin
6.     set(resynch, 0);
7.     loop
8.       block(timeout-event);
9.       switch (which-timer)
10.        case resynch:
11.           $i \leftarrow 1$ ;
12.          set(resynch,  $R_{synch}$ );
13.           $\Theta \leftarrow \text{clock}()$ ;
14.          broadcast("Time is  $\Theta$ ");
15.          set(transmit,  $W_{i,i+1}$ );
16.           $i \leftarrow i + 1$ ;
17.          break;
18.        case transmit:
19.           $\Theta \leftarrow \text{clock}()$ ;
20.          broadcast("Time is  $\Theta$ ");
21.          if ( $i < n$ ) set(transmit,  $W_{i,i+1}$ );
22.           $i \leftarrow i + 1$ ;
23.          break;
24.        end switch;
25.     end loop
26.   end

```

Figure 2.1 Clock synchronization process at the master node

```

1. process Slave(n:integer)
2.   var i: integer;
3.     T,R: array [1..n] of time;
4.   begin
5.     i ← 0;
6.     loop
7.       block(arrival of message "Time is Θ");
8.       i ← i+1;
9.       t ← clock();
10.      Ri ← t;
11.      Ti ← Θ;
12.      if (i = n) Then
13.        Test ← Rn -  $\frac{1}{n} \sum_{j=1}^n R_j$  +  $\frac{1}{n} \sum_{j=1}^n T_j$  +  $\bar{d}$ ;
14.        adjust-clock(Test);
15.        i ← 0;
16.      end if
17.    end loop
18.  end

```

Figure 2.2 Clock synchronization process at a slave node

The master process uses two timers, `resynch` and `transmit` (M.4). The `resynch` timer is used for measuring the resynchronization interval (M.12), while the `transmit` timer is used for measuring the interval ($W_{i,i+1}$) between the transmission of successive synchronization messages (M.15, M.21). The intervals $W_{i,i+1}$ have been left unspecified and may be chosen randomly. For example, the $W_{i,i+1}$ can be chosen to be identical and equal to some constant W . In Chapter 3, we analyze the algorithm under the more general assumption that the $W_{i,i+1}$ are i.i.d. random variables.

The function call `set(α , Δ)`, (M.6, M.12, M.15, M.21) sets the timer α to timeout after Δ units of time. The function call `block(event)` (M.8, S.7) causes the calling process to suspend until the specified event occurs. The call `broadcast(message)` (M.14, M.20), sends a copy of the message to all the slave nodes. Our algorithm does not require that synchronization messages be delivered in the order of transmission and hence the broadcast primitive does not have to preserve order. Even though the master process in Figure 2.1 deals with all the slave nodes simultaneously by using the broadcast call, it is not necessary for it to do so; the master process can deal with each slave separately, by using separate timers for each slave. The function call `clock()` (M.13, M.19, S.9) returns the current (logical) time at the node. The rest of the code is self-explanatory.

The number of synchronization messages (n) and the resynchronization interval (R_{synch}) are parameters of the algorithm. The values of these parameters are determined by the service specifications, i.e., by the desired maximum deviation between the clocks of the master and slave nodes and the desired probability of the actual deviation exceeding the specified maximum deviation (the probability of invalidity). The relationship between the service specification parameters and the parameters of the algorithm are discussed and deduced in Section 2.3 and Chapter 3.

2.3 Summary of Analysis

In this section, we provide an overview of the most important properties of TTP and the clock synchronization algorithm and illustrate the performance of

the algorithm with numerical examples. A detailed analysis and derivation of the properties is contained in Chapter 3.

2.3.1 Overview of Properties

The transmission error or deviation at resynchronization (ϵ) is later shown to be a random variable that depends both on the clock drift during the transmission period and the average end-to-end delay of the transmitted synchronization messages. However,

PROPERTY 1 *The magnitude of the average transmission error ($\bar{\epsilon}$) is negligible compared to the maximum transmission error, i.e.,*

$$|E[\epsilon]| \ll \epsilon_{\max},$$

if the separation between successive synchronization messages is i.i.d, the transmission period is short ($\frac{eT}{2} \ll \epsilon_{\max}$), and the expected value of the end-to-end delay is known accurately ($\Delta\bar{d} \ll \epsilon_{\max}$).

The transmission error or deviation at resynchronization (ϵ) has the following interesting property.

PROPERTY 2 *The probability distribution of the transmission error, ϵ approaches the Gaussian distribution $N\left(\bar{\epsilon}, \frac{\sigma_d^2}{n}\right)$, as the number, n , of synchronization messages increases.*

We refer to the minimum value of n for $N\left(\bar{\epsilon}, \frac{\sigma_d^2}{n}\right)$ to approximate $G_n(\epsilon)$, the probability distribution of ϵ , with an accuracy ξ as the *Gaussian cutoff* corresponding to ξ . We denote this cutoff value by n_g . In Chapter 3, we state why we believe that a $n_g < 10$ would be sufficient to approximate the distribution of ϵ by a Gaussian distribution with a good accuracy.

The following theorem provides an analytical expression for computing the number of synchronization messages required.

Table 2.1 erfc(x)

Complementary Error function			
x	erfc(x)	x	erfc(x)
0.000	1.0×10^0	3.459	1.0×10^{-6}
1.163	1.0×10^{-1}	3.766	1.0×10^{-7}
1.822	1.0×10^{-2}	4.052	1.0×10^{-8}
2.327	1.0×10^{-3}	4.320	1.0×10^{-9}
2.751	1.0×10^{-4}	4.572	1.0×10^{-10}
3.123	1.0×10^{-5}	4.812	1.0×10^{-11}

THEOREM 1 *The minimum number of messages required to guarantee a maximum deviation of ϵ_{max} with a probability of invalidity of p is given by $n_{min} = \max(n_g, n_e)$, where*

$$n_e = \frac{2\sigma_d^2(\text{erfc}^{-1}(p))^2}{\epsilon_{max}^2}, \quad (2.4)$$

whenever Property 1 holds. Here $\text{erfc}^{-1}(p)$ is the inverse of the complementary error function defined as $\text{erfc}(u) \triangleq 1 - \text{erf}(u)$, where $\text{erf}(u) \triangleq \frac{2}{\sqrt{\pi}} \int_0^u e^{-y^2} dy$.

The erfc function is a decreasing function of its argument and so it can be seen, by solving Eq. (2.4) for p , that the probability of invalidity decreases, for a given ϵ_{max} , with increasing n . The rate of decrease may be gauged by referring to Table 2.1. However, the following important result provides a clearer idea of the rate of decrease.

PROPERTY 3 *For a given ϵ_{max} ($\epsilon_{max} > |\bar{\epsilon}|$), the probability of invalidity, p , decreases exponentially or better with the number of messages, i.e., there exists a bounding function for p of the form $\alpha e^{-n\beta}$ such that the magnitude of α does not increase with n and $\beta > 0$ is independent of n , i.e., $p = \mathbf{P}[|\epsilon| > \epsilon_{max}] < \alpha e^{-n\beta}$.*

The following property describes the most important characteristic of our clock synchronization algorithm.

PROPERTY 4 *Our clock synchronization algorithm can guarantee (with an associated probability of invalidity of the guarantee) an upper bound, γ_{max} , on the deviation between any two clocks in the system i.e.,*

$$\forall i, j, t |L_i(t) - L_j(t)| \leq \gamma_{max},$$

where

$$\gamma_{max} = 2(\epsilon_{max} + \rho(R_{synch} + d_{max} - d_{min})), \quad (2.5)$$

i, j denote processes, and $L_q(t)$ denotes the time on process q 's logical clock at real time t .

2.3.2 Performance

To illustrate the magnitude of the maximum deviation between clocks achievable with our algorithm, we present two numerical examples below. The values used for various system parameters, such as statistics of the end-to-end message delay and clock drift rate, are based on data reported in [23]. The data reported in [23] are from an experiment involving a measurement of 5000 message round trip delays, between two light weight processes running on two IBM 4381 processors connected via a channel-to-channel local area network. The measured maximum and minimum round trip delays in this experiment were 93.17 millisecc and 4.22 millisecc, the average delay was 4.91 millisecc and the median delay was 4.48 millisecc. In addition, 95 % of all observed delays were shorter than 5.2 millisecc. On the basis of this last observation, we can estimate the standard deviation σ_d of the end-to-end delays to be 0.184 millisecc by assuming that the distribution of end-to-end delays can be approximated by a Gaussian distribution (this assumption is justified in Section 3.2.3). In making this estimate, we have made use of the fact that the deviation from the median for 95 % of the cases in the above round trip delay data (scaled down by a factor of two) is less than 0.36 millisecc (if the probability of the event that a normally distributed random variable differs from its median by more than 0.36 is 0.05, then the standard deviation of the variable is 0.184). For convenience, we use the slightly larger value of 0.2 for σ_d .

If we assume a ρ of 6 $\mu\text{sec}/\text{sec}$ and the desired probability of invalidity is 1×10^{-9} , then, by Property 4, we can guarantee a γ_{max} of 2 millisecc by choosing $\epsilon_{max} = 0.6$ millisecc and $\rho(R_{synch} + d_{max} - d_{min}) = 0.4$ millisecc. This gives an R_{synch} of 67 sec, and by Theorem 1 (or Table 2.2) $n = 10$ messages (assuming $n_g = 10$).

If a less tight γ_{max} of 4 millisecc is desired, then the resynchronization interval can be longer. For example, a resynchronization interval of 200 sec can be achieved by choosing $\rho(R_{synch} + d_{max} - d_{min}) = 1.2$ millisecc and $\epsilon_{max} = 0.8$ millisecc. Note that the number of synchronization messages required is 10 in both cases. This is because we have assumed that $n_g = 10$. Actually, a more realistic estimate of n_g would be much smaller, since the end-to-end message delays have a distribution that is approximately Gaussian (as will be seen in Section 3.2.3). Thus a smaller number of synchronization messages would actually be sufficient in these examples.

A caveat about the values derived above for the number of synchronization messages: it follows from Theorem 1 that, since $n_e \propto \left(\frac{\sigma_d}{\epsilon_{max}}\right)^2$, the number of synchronization messages is sensitive to the value that we assume for σ_d , when ϵ_{max} is of the same order as or smaller than σ_d ; thus, for example, if the standard deviation of end-to-end delays were actually 0.6 millisecc, instead of the 0.2 millisecc that we obtained using the Gaussian approximation, then the number of synchronization messages in the first example would be much larger than the value derived above (38 instead of 10).

2.3.3 Comparison

Deterministic Algorithms

The γ_{max} 's in these examples are better than the best γ_{max} of $\frac{1}{2}(d_{max} - d_{min}) \approx 50$ millisecc achievable with the clock synchronization algorithms described in [26], [31], [47], [48], [53], [54] and [63] which have to conform to Eq. (1.2). Also, by Property 3 it should be possible to reduce the probability of invalidity substantially by increasing the number of messages by a small number. This is illustrated in

Table 2.2 Number of synchronization messages versus p and $\frac{\epsilon_{max}}{\sigma_d}$ ($n_g = 10$)

n vs. p ($\frac{\epsilon_{max}}{\sigma_d} = 1$)		n vs. $\frac{\epsilon_{max}}{\sigma_d}$ ($p = 1 \times 10^{-9}$)	
p	n	$\frac{\epsilon_{max}}{\sigma_d}$	n
1.0×10^{-6}	24	10.0	10
1.0×10^{-7}	29	3.0	10
1.0×10^{-8}	33	2.0	10
1.0×10^{-9}	38	1.0	38
1.0×10^{-10}	42	0.75	68
1.0×10^{-11}	47	0.50	152

Table 2.2. Note that p can be reduced by an order of magnitude by increasing the number of synchronization messages by just 5.

Cristian's Algorithm

The γ_{max} 's achievable with our algorithm are comparable to those achievable with Cristian's probabilistic algorithm [23].

To guarantee a γ_{max} of 2 millisecc with a probability of failing to reach rapport of 1×10^{-9} , Cristian's algorithm requires that, on the average, about 4 messages be sent every 67 seconds [23]. Our algorithm requires 10 messages to be sent every 67 seconds to realize the same specifications. To guarantee a γ_{max} of 4 millisecc with the same probability of invalidity, Cristian's algorithm requires that about 2.1 messages be sent, on the average, every 231 seconds. Our algorithm requires 10 messages to be sent every 200 seconds. As we had pointed out earlier, the number of messages required by our algorithm would be smaller if we make use of a more realistic estimate of n_g . (The reader should also keep in mind the caveat that we had stated earlier: the validity of the comparison provided here depends on the actual value of the standard deviation of end-to-end delays being close to the value that we assumed in our examples).

There are two points to be noted in comparing our algorithm with Cristian's algorithm. First, in a distributed system based on a broadcast network like Ethernet, with our algorithm, the master node need broadcast only one set of synchronization messages per resynchronization interval. In Cristian's algorithm, since each slave is individually responsible for maintaining synchronization with the master node, as many sets of synchronization messages will have to be transmitted as there are slave nodes. Thus the number of messages required by Cristian's algorithm will have to be multiplied by a factor of k for a system with k slave nodes.

Second, our algorithm offers the possibility of piggybacking synchronization information onto regular messages thus saving on synchronization messages. Our algorithm does not make any assumptions about the interval of separation ($W_{i,i+1}$) between the transmissions of successive synchronization messages. This gives rise to the following interesting possibility. If a minimum level of message traffic always exists between M and S, so that we can assume that at least n messages, apart from the clock synchronization messages, are sent from M to S over a transmission period, then there is no need for special clock synchronization messages at all. M can stamp each message that it sends to S with the time on its clock, and these time-stamped messages now carry the synchronization information that S requires.

In addition, the following differences between our algorithm and Cristian's algorithm should be pointed out [24].

Cristian's algorithm guarantees an upper bound on the maximum deviation between the clocks of any two synchronized processes. However there is a non-zero probability that a process is not synchronized, i.e., a process fails to reach rapport. This probability may be made very small, for a specified upper bound on the deviation between the clocks, by increasing the number of attempts at rapport; Our algorithm guarantees that synchronized clocks are within a certain maximum deviation with a non-zero probability of the actual deviation exceeding the maximum. However this probability can be made as small as desired, by transmitting a sufficient number of synchronization messages. In Cristian's algorithm, a process knows when it is synchronized and when it is not, while in our algorithm a process cannot

know this. However the probability of loss of synchronization in both cases is so small that we do not expect to observe the event of loss of synchronization.

Cristian's algorithm maintains clocks within a specified maximum deviation for any message delay distribution whether dependent or independent of time. If the message delay distribution is dependent on time, then either loss of synchronization or improvement of reading error may occur. If network load increases, then an increase in the maximum deviation between clocks may have to be tolerated. Our algorithm guarantees that the probability that the deviation between two clocks at resynchronization will exceed a specified maximum deviation is smaller than a specified probability of invalidity, whatever be the distribution of message delays. If the expected value or variance of the delay distribution changes with time (and estimates of these moments are not made dynamically), then this probability may exceed the specified probability of invalidity or a smaller transmission error may be achievable with the specified probability of invalidity.

In Cristian's algorithm, the maximum deviation between two clocks at a resynchronization point is not dependent on any characteristic of the message delay distribution such as expected value or variance, and a bound on it is known to the slave process at the resynchronization point. In our algorithm, the slave does not know a bound on the maximum deviation between clocks at a resynchronization point, but a bound on the probability that the actual deviation will exceed a specified maximum deviation is known. Determining this bound on the probability requires a knowledge of the actual or worst case values of the expected value and variance of the message delay distribution.

2.4 Summary

In this chapter, we addressed the problem of reducing the maximum clock skew (thereby improving the resolution of the system-wide time base) that can be achieved in a distributed real-time system. Due to the constraints imposed by Eq. (1.2) on the maximum clock skew that can be guaranteed with certainty, we took a probabilistic

approach to the problem and developed a new probabilistic clock synchronization algorithm.

We presented a new time transmission protocol, TTP, that can achieve very small transmission errors, and described a probabilistic clock synchronization algorithm that incorporates this protocol. The time transmission protocol involves the transmission of multiple synchronization messages and works by smoothing out the random variations in the end-to-end delays of the synchronization messages through a process of averaging. The clock synchronization algorithm was constructed from TTP using a master-slave approach.

The clock synchronization algorithm presented, which is not subject to the bounds imposed by Eq. (1.2), was shown to perform much better than the deterministic algorithms that are constrained to obey Eq. (1.2). We demonstrated, in our examples, that our algorithm is capable of guaranteeing a maximum deviation between clocks of the order of a few milliseconds, compared to the smallest maximum deviation of 50 milliseconds that can be guaranteed by deterministic algorithms. However the guarantee offered by our algorithm is probabilistic, i.e., there is a non-zero probability that the guarantee offered by our algorithm will fail to hold. The probability of invalidity however can be made *extremely* small by transmitting a sufficient number of messages. In our examples, we were able to achieve a probability of invalidity of (1×10^{-9}) by transmitting 10 synchronization messages. We also showed that the performance of our algorithm is comparable to the performance of the probabilistic algorithm proposed by Cristian [23]. Further unlike Cristian's algorithm in which each slave is individually responsible for synchronizing with the master node, in our algorithm (in which the master is responsible for getting the slaves to synchronize with itself) the number of messages required to achieve a required maximum clock skew can be reduced by a factor equal to the number of nodes in the system, if a broadcast channel is used.

We also presented a selective overview of some properties exhibited by the clock synchronization protocol. In the next chapter, we provide a detailed analysis of the

protocol and derive various properties exhibited by the protocol, including various bounds on the maximum clock skew.

CHAPTER 3

ANALYSIS OF OUR CLOCK SYNCHRONIZATION PROTOCOL

This chapter contains a detailed analysis of our clock synchronization algorithm. The three most important variables of our algorithm are the deviation between clocks, the probability of invalidity and the number of synchronization messages. We first derive expressions for the deviation at resynchronization or the transmission error (ϵ) and its first and second moments, in Section 3.1. Next, in Section 3.2, we derive various bounds on the probability of invalidity expressed in terms of these moments. In Section 3.3, we relate these moments to the number of synchronization messages and deduce a relationship between the number of messages, the probability of invalidity and the maximum deviation at resynchronization. We also show that the probability of invalidity decreases exponentially or better with the number of messages. We provide a discussion of various aspects of the clock synchronization algorithm in Section 3.5. We conclude the chapter in Section 3.6 with a summary and identification of potential directions for future research.

3.1 Transmission Error

LEMMA 1 *Let d_i and δ_i respectively denote the end-to-end delay of the i th message and the deviation between the clocks of S and M when the i th message is received at S . The transmission error ϵ is given by:*

$$\epsilon = \Delta\delta_n - \Delta\bar{d}(n) \tag{3.1}$$

where $\Delta\delta_n \triangleq \delta_n - \bar{\delta}(n)$, $\Delta\bar{d}(n) \triangleq \bar{d}(n) - \bar{d}$, $\bar{\delta}(n) = \frac{1}{n} \sum_{i=1}^n \delta_i$ and $\bar{d}(n) = \frac{1}{n} \sum_{i=1}^n d_i$.

PROOF: The i th message is received at S d_i time units after it was stamped with the time on it. When it arrives at S, the clock of S is ahead of the clock of M by δ_i time units. Hence

$$R_i = T_i + d_i + \delta_i$$

and

$$\bar{R}(n) = \bar{T}(n) + \bar{d}(n) + \bar{\delta}(n).$$

The latter equation can be rewritten as

$$\bar{T}(n) - \bar{R}(n) = -(\bar{d}(n) + \bar{\delta}(n)). \quad (3.2)$$

The n th message was sent at time T_n on M's clock and it took d_n units of time to arrive at S. Hence the actual time (T_{act}) on M's clock when the n th message is received by S is equal to the sum of the time stamped on the message and its delay. Alternatively, T_{act} may be obtained by deducting the deviation (δ_n) between S and M at the time of arrival of the n th message at S, from the time of arrival. Hence

$$T_{act} = T_n + d_n = R_n - \delta_n. \quad (3.3)$$

The transmission error is then given by equations (2.2), (2.3) and (3.3) as

$$\begin{aligned} \epsilon &= T_{est} - T_{act} \\ &= (R_n - \bar{R}(n) + \bar{T}(n) + \bar{d}) - (R_n - \delta_n) \\ &= \bar{d} + \delta_n - (\bar{d}(n) + \bar{\delta}(n)) \quad (\text{from Eq. (3.2)}) \\ &= (\delta_n - \bar{\delta}(n)) - (\bar{d}(n) - \bar{d}) \\ &= \Delta\delta_n - \Delta\bar{d}(n) \end{aligned}$$

Q.E.D.

We make use of the following assumption to derive an useful approximation for ϵ in the next lemma:

$$n = o\left(\frac{1}{\rho}\right), \quad (3.4)$$

i.e., the number of messages is much smaller than $\frac{1}{\rho}$ (which is of the order of 10^6).

LEMMA 2 Let δ_0 denote the deviation between the clocks of S and M at the start of transmission of msg_1 , δ_i^r denote the increase in deviation between the clocks between the times of transmission of msg_1 and msg_i , and δ_i^d denote the increase in deviation during the time it takes msg_i to reach S after it has been transmitted (i.e., the increase in deviation during the message delay d_i of msg_i). Under the assumption that Eq. (3.4) holds, ϵ can be approximated as

$$\epsilon = \Delta\delta_n^r - \Delta\bar{d}(n) \quad (3.5)$$

where $\Delta\delta_n^r = \delta_n^r - \bar{\delta}_r(n)$ and $\bar{\delta}_r(n) = \frac{1}{n} \sum_{i=1}^n \delta_i^r$.

PROOF: The quantity δ_i represents the amount by which the clock of S is ahead of the clock of M when the i th message arrives at S. It can be decomposed into three components, viz., the deviation between the clocks of S and M at the start of transmission of msg_1 (δ_0), the increase in deviation between the clocks of S and M between the times of transmission of msg_1 and msg_i (δ_i^r), and δ_i^d , the increase in deviation during the time it takes message msg_i to reach S after it has been transmitted (δ_i^d). Thus,

$$\delta_i = \delta_0 + \delta_i^r + \delta_i^d \quad (1 \leq i \leq n).$$

Hence,

$$\bar{\delta}(n) = \frac{1}{n} \sum_{i=1}^n \delta_i = \delta_0 + \bar{\delta}_r(n) + \frac{1}{n} \sum_{i=1}^n \delta_i^d.$$

Therefore,

$$\Delta\delta_n = \delta_n - \bar{\delta}(n) = \delta_n^r - \bar{\delta}_r(n) + \delta_n^d - \frac{1}{n} \sum_{i=1}^n \delta_i^d. \quad (3.6)$$

Each δ_i^d can be written as $\delta_i^d = \hat{\rho}d_i$, where $\hat{\rho} \leq \rho$ is the actual relative drift rate. Eq. (3.6) can then be rewritten as,

$$\Delta\delta_n = \delta_n^r - \bar{\delta}_r(n) + \hat{\rho}d_n - \frac{1}{n} \sum_{i=1}^n \hat{\rho}d_i. \quad (3.7)$$

It follows from Lemma 1 that

$$\begin{aligned}
\epsilon &= \Delta\delta_n - \Delta\bar{d}(n) \\
&= \delta_n^\tau - \bar{\delta}_\tau(n) + \hat{\rho}d_n - \frac{1}{n} \sum_{i=1}^n \hat{\rho}d_i - \frac{1}{n} \sum_{i=1}^n d_i + \bar{d} \\
&= \delta_n^\tau - \bar{\delta}_\tau(n) - \frac{1}{n} \sum_{i=1}^n d_i + \bar{d} \\
&= \delta_n^\tau - \bar{\delta}_\tau(n) - \Delta\bar{d}(n) \\
&= \Delta\delta_n^\tau - \Delta\bar{d}(n).
\end{aligned}$$

In the third step above, we made use of Eq. (3.4) and neglected $\hat{\rho}$ and $\frac{\hat{\rho}}{n}$ in comparison to $\frac{1}{n}$.

Q.E.D.

LEMMA 3

$$\Delta\delta_n^\tau = \frac{\hat{\rho}}{n} \sum_{i=1}^{n-1} iW_{i,i+1},$$

where $\hat{\rho}$ denotes the actual relative drift rate between the clocks of M and S during the transmission period and $W_{i,j}$ is the separation between the times of transmission of messages msg_i and msg_j .

PROOF: The increase in the deviation between the clocks of M and S between the times of transmission of messages msg_1 and msg_i , δ_i^τ is given by

$$\begin{aligned}
\delta_i^\tau &= \hat{\rho}W_{1,i} \\
&= \hat{\rho} \sum_{j=1}^{i-1} W_{j,j+1}.
\end{aligned}$$

Therefore,

$$\bar{\delta}_\tau(n) = \frac{1}{n} \sum_{i=1}^n \delta_i^\tau$$

$$\begin{aligned}
&= \frac{1}{n} \sum_{i=1}^n \hat{\rho} \sum_{j=1}^{i-1} W_{j,j+1} \\
&= \frac{\hat{\rho}}{n} \sum_{i=1}^n \sum_{j=1}^{i-1} W_{j,j+1} \\
&= \frac{\hat{\rho}}{n} \sum_{i=1}^{n-1} (n-i) W_{i,i+1}
\end{aligned}$$

and

$$\begin{aligned}
\Delta \delta_n^\tau &= \delta_n^\tau - \bar{\delta}_\tau(n) \\
&= \hat{\rho} \sum_{i=1}^{n-1} W_{i,i+1} - \frac{\hat{\rho}}{n} \sum_{i=1}^{n-1} (n-i) W_{i,i+1} \\
&= \frac{\hat{\rho}}{n} \sum_{i=1}^{n-1} n W_{i,i+1} - \frac{\hat{\rho}}{n} \sum_{i=1}^{n-1} (n-i) W_{i,i+1} \\
&= \frac{\hat{\rho}}{n} \sum_{i=1}^{n-1} i W_{i,i+1}.
\end{aligned}$$

Q.E.D.

From Lemmas 1,2 and 3, we can write ϵ as

$$\epsilon = \frac{\hat{\rho}}{n} \sum_{i=1}^{n-1} i W_{i,i+1} - \frac{1}{n} \sum_{i=1}^n d_i + \bar{d}. \quad (3.8)$$

Note that ϵ is the sum of several random variables and hence is itself a random variable.

If the $W_{i,i+1}$ are chosen to be i.i.d., then ϵ has an expected value ($\bar{\epsilon}$) given by

$$E[\epsilon] = \frac{\hat{\rho} E[\tau]}{2} - \frac{1}{n} \sum_{i=1}^n (E[d_i] - \bar{d}). \quad (3.9)$$

This follows from Eq. (3.8) and the fact that

$$\tau = \sum_{i=1}^{n-1} W_{i,i+1},$$

and

$$\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}.$$

Let

$$\Delta \bar{d} \triangleq \max \{ |E[d_i] - \bar{d}| \}.$$

$\Delta \bar{d}$ represents the maximum error in the estimate, \bar{d} , of the expected value of the end-to-end delay used in TTP. Clearly,

$$E[\epsilon] \leq \frac{\rho E[\tau]}{2} + \Delta \bar{d}. \quad (3.10)$$

If the synchronization messages are separated by constant intervals of time ($W_{i,i+1} = W = \frac{\tau}{n-1}$), then Eq. (3.10) reduces to

$$E[\epsilon] \leq \frac{\rho(n-1)W}{2} + \Delta \bar{d}.$$

In [7], we show that $E[\epsilon] \leq \rho\tau$ (assuming $\Delta \bar{d} = 0$), even when the $W_{i,i+1}$ are not i.i.d.

Note that,

PROPERTY 1 *If the separation between successive synchronization messages is i.i.d., then the magnitude of the average transmission error ($\bar{\epsilon}$) is negligible compared to the maximum transmission error, i.e.,*

$$|E[\epsilon]| \ll \epsilon_{\max},$$

if the transmission period is short ($E[\tau] \ll \frac{2\epsilon_{\max}}{\rho}$), and the expected value of the end-to-end delay is known accurately ($\Delta \bar{d} \ll \epsilon_{\max}$).

The variance of ϵ is given by

$$V[\epsilon] \leq \frac{(n-1)(2n-1)}{6n} \rho^2 V[W_{i,i+1}] + \frac{1}{n^2} \sum_{i=1}^n V[d_i], \quad (3.11)$$

if the interval between the transmission of successive synchronization messages is i.i.d., and the end-to-end delays of synchronization messages are independent of each other. This follows from Eq. (3.8) and the fact that

$$\sum_{i=1}^{n-1} i^2 = \frac{(n-1)n(2n-1)}{6}.$$

Let σ_d^2 denote the worst case (maximum) estimate of the variance of the end-to-end delay. Eq. (3.11) then reduces to

$$V[\epsilon] \leq \frac{(n-1)(2n-1)}{6n} \rho^2 V[W_{i,i+1}] + \frac{\sigma_d^2}{n}. \quad (3.12)$$

Thus the better the estimate of the variance of the end-to-end delay, the smaller is the bound on the variance of the transmission error.

Eq. (3.12) reduces to

$$V[\epsilon] \leq \frac{\sigma_d^2}{n}, \quad (3.13)$$

if the $W_{i,i+1}$ are identical. It can be shown [7] that, in general, the variance is given by Eq. (3.13), whenever the transmission period is sufficiently short ($\rho^2 \tau^2 \ll \kappa \frac{\epsilon_{max}}{\rho}$, where $\kappa = \min\left(1, \frac{1}{\sqrt{2} \operatorname{erfc}^{-1}(p)}\right)$ and p is the desired probability of invalidity). We assume in the rest of the chapter that the conditions for the satisfaction of Eq. (3.13) hold.

3.2 Probability of Invalidity

In this section, we analyze the probability of invalidity, the probability that the transmission error or the deviation at resynchronization exceeds a specified maximum value. We consider four different bounds on the probability of invalidity, viz., the *experimental*, *Tchebycheff*, *error function* and *exponential* bounds. The last two of the bounds are derived on the assumption that the end-to-end delays of synchronization messages are independent.

3.2.1 Experimental Bound

The probability of invalidity or a bound on it may be deduced empirically. The difficulty with this approach is that the probability of invalidity that we are seeking is so small ($\approx 10^{-9}$) that the number of trials of the experiment will have to be very large.

3.2.2 Tchebycheff Bound

Alternatively, an upper bound on the probability of invalidity may be determined using Tchebysheff's inequality. Tchebycheff's inequality states that

$$\mathbf{P} [|\epsilon - \bar{\epsilon}| \geq \Delta\epsilon] \leq \frac{V[\epsilon]}{(\Delta\epsilon)^2}$$

for any $\Delta\epsilon > 0$. The following result is useful in determining the probability of invalidity from Tchebycheff's inequality.

LEMMA 4

$$\mathbf{P} [|\epsilon| > \epsilon_{max}] \leq \mathbf{P} [|\epsilon - \bar{\epsilon}| > \Delta\epsilon_{max}],$$

if $\Delta\epsilon_{max} > 0$, where $\Delta\epsilon_{max} = \epsilon_{max} - |\bar{\epsilon}|$.

PROOF: This result follows from the observation that irrespective of whether $\bar{\epsilon}$ is positive or negative, the range of values of ϵ , $[-\Delta\epsilon_{max} + \bar{\epsilon}, \bar{\epsilon} + \Delta\epsilon_{max}]$, is a subrange of the range of values, $[-|\bar{\epsilon}| - \Delta\epsilon_{max}, |\bar{\epsilon}| + \Delta\epsilon_{max}]$. This implies that

$$\mathbf{P} [-|\bar{\epsilon}| - \Delta\epsilon_{max} \leq \epsilon \leq |\bar{\epsilon}| + \Delta\epsilon_{max}] \geq \mathbf{P} [-\Delta\epsilon_{max} + \bar{\epsilon} \leq \epsilon \leq \bar{\epsilon} + \Delta\epsilon_{max}],$$

i.e.,

$$\mathbf{P} [|\epsilon| \leq \epsilon_{max}] \geq \mathbf{P} [|\epsilon - \bar{\epsilon}| \leq \Delta\epsilon_{max}],$$

or equivalently

$$\mathbf{P} [|\epsilon - \bar{\epsilon}| \geq \Delta\epsilon_{max}] \geq \mathbf{P} [|\epsilon| \geq \epsilon_{max}],$$

from which the result follows.

Q.E.D.

From Lemma 4 and Tchebycheff's inequality, we can compute the following upper bound on the probability of invalidity,

$$P[|\epsilon| \geq \epsilon_{max}] \leq \frac{V[\epsilon]}{(\epsilon_{max} - |\bar{\epsilon}|)^2}. \quad (3.14)$$

We refer to this bound as the *Tchebycheff bound*.

The Tchebycheff bound on the probability of invalidity is proportional to the variance of the transmission error. If we wish to reduce this bound on the probability of invalidity, we can try to reduce the variance of the transmission error. However the corresponding decrease in the bound will be *linear*.

3.2.3 Gaussian Approximation Bounds

In the previous section, we established a bound on the probability of invalidity using Tchebysheff's inequality. Tchebysheff's inequality holds generally for any probability distribution and the upper bound that it provides may be conservative for specific probability distributions. This statement is especially true of the Gaussian distribution, which is of particular interest to us since the distribution of ϵ tends to become Gaussian for large n . This is proved in the following theorem. The validity of this theorem requires that the end-to-end delays of synchronization messages be independent of each other.

PROPERTY 2 *If the end-to-end delays of synchronization messages are independent, the probability distribution of the transmission error (ϵ) approaches the Gaussian distribution $N(\bar{\epsilon}, V[\epsilon])$, as the number of messages n increases.*

PROOF: As can be seen from Eq. (3.8), ϵ is the sum of several independent random variables. The result follows from the central limit theorem.

Q.E.D.

This result can be rewritten as follows. Let $G_n(\epsilon)$ denote the probability distribution function of ϵ , $\Phi(\epsilon)$ denote the Gaussian probability distribution function given by $\Phi(\epsilon) \triangleq \frac{1}{\sqrt{2\pi\sigma}} \int_{-\infty}^{\epsilon} e^{-\frac{1}{2}\left(\frac{t-\bar{\epsilon}}{\sigma}\right)^2} dt$ and $\sigma \triangleq \sqrt{(V[\epsilon])}$. Then for all $n > n_g$,

$$\left| \frac{G_n(\epsilon) - \Phi(\epsilon)}{\Phi(\epsilon)} \right| < \xi$$

where $\xi > 0$ can be made arbitrarily small by choosing n_g large enough. As mentioned in Section 2.3, we refer to n_g , the minimum value of n for $N(\bar{\epsilon}, \sigma^2)$ to approximate $G_n(\epsilon)$ with an accuracy ξ as the *Gaussian cutoff*. If d itself has an approximately Gaussian distribution, then the Gaussian cutoff would be close to 1. If the distribution of d is not close to Gaussian, then n_g would be larger and has to be determined by experiment. In a simulation study of the relationship between n_g and ξ , in which we assumed an uniform distribution for d and $\tau = 0$, we found that an $n_g > 5$ results in an error due to approximation of less than 5% within the first standard deviation, less than 3% between the first and second standard deviations and less than 0.5% beyond the second standard deviation. Thus an $n_g > 5$ (say $n_g = 10$, to be safe) would be sufficient to approximate the distribution of ϵ by a Gaussian distribution with a good accuracy, even when the distribution of d is uniform (i.e., not close to Gaussian). But we conjecture that the distribution of d would be approximately Gaussian and hence a smaller n_g would suffice. Our conjecture is based on the fact that the end-to-end message delay d itself is the sum of several independent random delays [47]:

1. *Send time*: the variable time required by the sender to assemble and send a synchronization message
2. *Access-time*: the variable access time of the sender to the communication medium, depending on the access strategy.
3. *Propagation delay*: the variable propagation delay of the message depending on distance and channel.

4. *Receive time*: the variable time required to switch contexts and schedule the receiving process to run at the receiving node and the time to process the message.

Hence by the central limit theorem the distribution of d itself is likely to be somewhat Gaussian.

Error Function Bound

We next derive a bound on the probability of invalidity, under the assumption that Property 2 holds true, i.e., under the assumption that the end-to-end delays of synchronization messages are independent and the number of synchronization messages is greater than n_g .

LEMMA 5

$$\mathbf{P} [|\epsilon| \geq \epsilon_{max}] \leq \text{erfc} \left(\frac{\Delta \epsilon_{max}}{\sqrt{2V[\epsilon]}} \right)$$

if $\Delta \epsilon_{max} > 0$. Here $\Delta \epsilon_{max} = \epsilon_{max} - |\bar{\epsilon}|$, and $\text{erfc}(u)$ is the complementary error function defined as $\text{erfc}(u) \triangleq 1 - \text{erf}(u)$, where

$$\text{erf}(u) \triangleq \frac{2}{\sqrt{\pi}} \int_0^u e^{-y^2} dy.$$

PROOF: For a normally distributed random variable X , with distribution $N(\bar{X}, \sigma_X^2)$, the probability that X lies within ω of its mean \bar{X} is given by

$$\begin{aligned} \mathbf{P} [|X - \bar{X}| \leq \omega] &= \int_{-\omega + \bar{X}}^{\omega + \bar{X}} \frac{1}{\sqrt{2\pi}\sigma_X} e^{-\frac{1}{2} \left(\frac{x - \bar{X}}{\sigma_X} \right)^2} dx \\ &= 2 \int_{\bar{X}}^{\omega + \bar{X}} \frac{1}{\sqrt{2\pi}\sigma_X} e^{-\frac{1}{2} \left(\frac{x - \bar{X}}{\sigma_X} \right)^2} dx \\ &= \frac{2}{\sqrt{\pi}} \int_0^{\frac{\omega}{\sqrt{2}\sigma_X}} e^{-y^2} dy \\ &= \text{erf} \left(\frac{\omega}{\sqrt{2}\sigma_X} \right) \end{aligned}$$

Therefore,

$$\mathbf{P} \left[|X - \bar{X}| \geq \omega \right] = 1 - \operatorname{erf} \left(\frac{\omega}{\sqrt{2}\sigma_X} \right) = \operatorname{erfc} \left(\frac{\omega}{\sqrt{2}\sigma_X} \right) \quad (3.15)$$

For $n > n_g$ we can approximate $G_n(\epsilon)$ by $N(\bar{\epsilon}, V[\epsilon])$. Hence by Eq. (3.15)

$$\mathbf{P} \left[|\epsilon - \bar{\epsilon}| \geq \Delta\epsilon_{max} \right] = \operatorname{erfc} \left(\frac{\Delta\epsilon_{max}}{\sqrt{2V[\epsilon]}} \right) \quad (3.16)$$

By Lemma 4 and Eq. (3.16) we have,

$$\mathbf{P} \left[|\epsilon| \geq \epsilon_{max} \right] \leq \mathbf{P} \left[|\epsilon - \bar{\epsilon}| \geq \Delta\epsilon_{max} \right] = \operatorname{erfc} \left(\frac{\Delta\epsilon_{max}}{\sqrt{2V[\epsilon]}} \right)$$

Q.E.D.

We refer to the upper bound on the probability of invalidity derived above as the *error function bound*. The error function bound depends on the variance of the transmission error through the complementary error function. As we saw earlier, the complementary error function is extremely sensitive to its argument. Thus by decreasing the variance of the transmission error by a small amount, the bound on the probability of invalidity can be reduced by orders of magnitude. The reader should compare the dependence of this bound on the variance of the transmission error, with that of the bound derived in Section 3.2.2 using Tchebysheff's inequality.

Exponential Bound

The following bound, also derived using the Gaussian approximation, shows the exponential nature of the dependence of the probability of invalidity on the variance of the transmission error.

LEMMA 6

$$\mathbf{P} [|\epsilon| \geq \epsilon_{max}] < \sqrt{\frac{2V[\epsilon]}{\pi}} \frac{e^{-\frac{(\Delta\epsilon_{max})^2}{2V[\epsilon]}}}{\Delta\epsilon_{max}},$$

if $\Delta\epsilon_{max} > 0$, where $\Delta\epsilon_{max} = \epsilon_{max} - |\bar{\epsilon}|$.

PROOF: Let $\sigma = \sqrt{V[\epsilon]}$. By Property 2, ϵ is distributed $N(\bar{\epsilon}, \sigma)$. Therefore by Lemma 4,

$$\begin{aligned} \mathbf{P} [|\epsilon| \geq \epsilon_{max}] &\leq \mathbf{P} [|\epsilon - \bar{\epsilon}| \geq \Delta\epsilon_{max}] \\ &= \frac{2}{\sqrt{2\pi}\sigma} \int_{\bar{\epsilon} + \Delta\epsilon_{max}}^{\infty} e^{-\frac{(t-\bar{\epsilon})^2}{2\sigma^2}} dt \\ &= \sqrt{\frac{2}{\pi}} \int_{\frac{\Delta\epsilon_{max}}{\sigma}}^{\infty} e^{-\frac{y^2}{2}} dy \end{aligned} \quad (3.17)$$

Let

$$Q(z) \triangleq \int_z^{\infty} e^{-\frac{y^2}{2}} dy \quad (3.18)$$

A bound can be derived [106] for $Q(z)$ as follows. For $z > 0$,

$$\begin{aligned} Q(z) &= \int_z^{\infty} \frac{1}{y} \left(ye^{-\frac{y^2}{2}} \right) dy \\ &= -\frac{1}{y} e^{-\frac{y^2}{2}} \Big|_z^{\infty} - \int_z^{\infty} \frac{e^{-\frac{y^2}{2}}}{y^2} dy \\ &= \frac{1}{z} e^{-\frac{z^2}{2}} - \int_z^{\infty} \frac{e^{-\frac{y^2}{2}}}{y^2} dy \end{aligned}$$

The second term in the last equation is always positive. Hence

$$Q(z) < \frac{1}{z} e^{-\frac{z^2}{2}} \quad (3.19)$$

From equations (3.17), (3.18) and (3.19), it follows that

$$\mathbf{P} [|\epsilon| \geq \epsilon_{max}] \leq \sqrt{\frac{2}{\pi}} Q\left(\frac{\Delta\epsilon_{max}}{\sigma}\right) < \sqrt{\frac{2}{\pi}} \frac{\sigma e^{-\frac{(\Delta\epsilon_{max})^2}{2\sigma^2}}}{(\Delta\epsilon_{max})}$$

The result follows by substituting $\sigma = \sqrt{V[\epsilon]}$.

Q.E.D.

We refer to the bound derived above as the *exponential bound* on the probability of invalidity.

3.3 Number of Messages

In this section, we make use of the bounds derived in Section 3.2, to derive an expression that can be used to compute the number of synchronization messages required to guarantee a specified maximum transmission error and probability of invalidity. We also show that the probability of invalidity decreases exponentially or better with the number of synchronization messages.

THEOREM 1 *The minimum number of messages required to guarantee a maximum deviation of ϵ_{max} with a probability of invalidity of p is given by $n_{min} = \max(n_g, n_e)$, where*

$$n_e = \frac{2\sigma_d^2(\text{erfc}^{-1}(p))^2}{(\Delta\epsilon_{max})^2}, \quad (3.20)$$

provided $\Delta\epsilon_{max} > 0$, where $\Delta\epsilon_{max} = \epsilon_{max} - |\bar{\epsilon}|$.

PROOF: By Lemma 5, we have

$$\mathbf{P} [|\epsilon| > \epsilon_{max}] \leq \text{erfc} \left(\frac{\Delta\epsilon_{max}}{\sqrt{2V[\epsilon]}} \right).$$

To guarantee a maximum deviation of ϵ_{max} with a probability of invalidity p , we must have $\mathbf{P} [|\epsilon| > \epsilon_{max}] \leq p$, i.e.,

$$\text{erfc} \left(\frac{\Delta\epsilon_{max}}{\sqrt{2V[\epsilon]}} \right) \leq p,$$

i.e.,

$$\operatorname{erfc}\left(\frac{\sqrt{n}\epsilon_{max}}{\sqrt{2}\sigma_d}\right) \leq p,$$

which when solved for n yields the n_e of Eq. (3.20). Since $n > n_g$ for the Gaussian approximation to be accurate, $n_{min} = \max(n_g, n_e)$.

Q.E.D.

Note that, whenever Property 1 holds, $\bar{\epsilon}$ can be neglected in comparison to ϵ_{max} . In this case, the minimum number of synchronization messages required can be determined in a straightforward manner, by replacing $\Delta\epsilon_{max}$ in Eq. (3.20) with ϵ_{max} .

We prove next that the probability of invalidity decreases exponentially or better with the number of messages.

PROPERTY 3 *For a given ϵ_{max} ($\epsilon_{max} > |\bar{\epsilon}|$), the probability of invalidity, p , decreases exponentially or better with the number of messages, i.e., there exists a bounding function for p of the form $\alpha e^{-n\beta}$ such that the magnitude of α does not increase with n and $\beta > 0$ is independent of n , i.e.,*

$$p = \mathbf{P} [|\epsilon| \geq \epsilon_{max}] < \alpha e^{-n\beta},$$

whenever Property 1 holds true.

PROOF: By Lemma 6,

$$\mathbf{P} [|\epsilon| \geq \epsilon_{max}] < \sqrt{\frac{2}{\pi}} \frac{\sigma_d e^{-\frac{n(\Delta\epsilon_{max})^2}{2\sigma_d^2}}}{\sqrt{n}\Delta\epsilon_{max}} \approx \sqrt{\frac{2}{\pi}} \frac{\sigma_d e^{-\frac{n\epsilon_{max}^2}{2\sigma_d^2}}}{\sqrt{n}\epsilon_{max}} \simeq \alpha e^{-n\beta}$$

We neglected $\bar{\epsilon}$ in comparison to ϵ_{max} , using Property 1, in the above equation.

Q.E.D.

Note that the coefficient α is a decreasing function of n ($\propto \frac{1}{\sqrt{n}}$). Thus the coefficient is also responsible for a decrease, albeit a weak one, in the probability of invalidity with increasing n .

3.4 Bound on the Deviation between Clocks

PROPERTY 4 *Our clock synchronization algorithm can guarantee (with an associated probability of invalidity of the guarantee) an upper bound, γ_{max} , on the deviation between any two clocks in the system i.e.,*

$$\forall i, j, t |L_i(t) - L_j(t)| \leq \gamma_{max},$$

where

$$\gamma_{max} = 2(\epsilon_{max} + \rho(R_{synch} + d_{max} - d_{min})), \quad (3.21)$$

i, j denote processes and $L_q(t)$ denotes the time on process q 's logical clock at real time t .

PROOF: TTP guarantees that the maximum deviation between the master and a slave immediately after a resynchronization is no more than ϵ_{max} with a probability of invalidity of p . The durations between successive resynchronizations at a slave can never exceed $R_{synch} + d_{max} - d_{min}$, since the worst case occurs when a slave receives the last synchronization message with minimum delay (i.e., at time $T_1 + \tau + d_{min}$, where T_1 is the time of transmission of the first synchronization message) in one synchronization cycle and with maximum delay (i.e., at time $T_1 + R_{synch} + \tau + d_{max}$) in the next cycle and for this case the duration between successive resynchronizations is given by $R_{synch} + d_{max} - d_{min}$. Thus the maximum deviation (due to clock drift) that can develop between the clocks of the master and a slave during the interval between two successive resynchronizations is given by $\rho(R_{synch} + d_{max} - d_{min})$. The maximum deviation between a slave and the master at any time is therefore given by:

$$\gamma_{ms}^{max} = \epsilon_{max} + \rho(R_{synch} + d_{max} - d_{min}) \quad (3.22)$$

The maximum deviation between any two clocks in the system is given by $2\gamma_{ms}^{max}$:

$$\gamma_{max} = 2\gamma_{ms}^{max} = 2(\epsilon_{max} + \rho(R_{synch} + d_{max} - d_{min}))$$

Q.E.D.

We have assumed in Eq. (3.21) that the distribution of the end-to-end message delay between the master process and every slave process is the same. It is possible to extend it to the case where the distributions are different for different slave processes, but we do not consider it further here.

3.5 Discussion

3.5.1 Independence

We assumed, in our analysis, that the end-to-end delays of synchronization messages are independent. This assumption enabled us to analytically derive the minimum number of synchronization messages required to guarantee a specified maximum transmission error and probability of invalidity. We do not know how these three quantities will be related and consequently how to guarantee a specified probability of invalidity and a specified maximum transmission error if this assumption does not hold. The relationship will have to be determined through experimentation. In this section, we look at one way of realizing the independence assumption.

Independence between successive synchronization messages can be achieved by ensuring that there is a minimum gap of W time units between the transmissions of the synchronization messages. This gap is chosen such that any transient network traffic bursts that may introduce a correlation between the delays experienced by successive synchronization messages die down within W time units with high probability. For example, Cristian in the examples discussed in [22] uses a separation time between successive attempts of 2 seconds to ensure independence of successive attempts (Cristian assumes independence of successive attempts in [22] in order to be able to analytically derive the average number of messages needed for synchronization and the probability of losing synchronization. The actual deviation between clocks immediately after a resynchronization is not dependent on this assumption. However if this assumption does not hold, it may not be possible to determine the probability that this resynchronization will actually occur and the

maximum number of attempts that will have to be permitted, without resorting to experimentation).

3.5.2 Moments of the End-to-End Delay

We assumed that estimates of the expected value and the variance of the end-to-end delays of synchronization messages during a transmission period are known. If a steady state can be assumed for the system, then these moments can be determined once statically and the same values used after that. However if a steady state cannot be assumed, i.e., the expected value and the standard deviation can change with time, then the worst case values for these quantities, can be used. However, this will have the effect of increasing the smallest ϵ_{max} to which our analysis is applicable (since many of our results are applicable only when $\Delta\epsilon_{max} > 0$), and increasing the number of synchronization messages (this number depends on the variance). Alternatively these quantities can be evaluated dynamically using one of the two techniques discussed below.

Meta-Level Process

A meta-level process can be used to keep track of the current values of the expected value and standard deviation of the end-to-end delays. Either a process in the node hosting the master process can be made responsible for this, or a special process in each slave node can be handle this. This scheme would be suitable for systems in which the delay statistics change very slowly.

Continuous Monitoring

The dependence of the transmission error on the expected value of the end-to-end delay can be eliminated by making a simple modification to the time transmission protocol. When the node S receives a synchronization message msg_i from the node M, it not only records the time of receipt of the message, but also returns the message to M immediately. On receipt of the returned message, M records the time

of receipt and computes the round trip time r_i of the synchronization message, i.e., the length of the interval between the time that the synchronization message was sent out and the time that it was received back. At the end of the transmission period, M sends the average $\bar{r}(n) = \frac{1}{n} \sum_{i=1}^n r_i$ of these round trip delays to S. On receipt of this information, S estimates the time on M's clock as:

$$T_{est} = R_n - \bar{R}(n) + \bar{T}(n) + \frac{1}{2}\bar{r}(n).$$

In a broadcast network, only one slave need be involved in this dynamic computation of the average. Once the master has computed $\bar{r}(n)$ on the basis of the messages returned by this slave, it can broadcast the value to the other slaves.

It can be shown that the transmission error ϵ is given, in this case, by

$$\epsilon = \frac{\hat{\rho}}{n} \sum_{i=1}^{n-1} iW_{i,i+1} - \frac{1}{n} \sum_{i=1}^n d_i + \frac{1}{2n} \sum_{i=1}^n r_i,$$

where d_i and r_i denote the end-to-end delay and round trip delay of the i th synchronization message. The expected value of the transmission error is then given by

$$E[\epsilon] = \frac{\hat{\rho}E[\tau]}{2}$$

assuming the separation interval between successive synchronization messages is i.i.d., and $E[r_i] = 2E[d_i]$. (The reader should compare the last three equations with Eq. (2.3), Eq. (3.8) and Eq. (3.9)).

Note that, the expected value of ϵ and, hence, the smallest ϵ_{max} to which our analysis is applicable, do not depend on the expected value of the end-to-end delay of synchronization messages any longer. The variance of the transmission error now depends on the variance of the difference between the upward and downward components of a round trip delay. We can compute the number of synchronization messages required to guarantee a specified probability of invalidity by using a worst case estimate of this variance. Thus, the requirement that the average and variance of the end-to-end delays be explicitly known, can be eliminated using this method.

3.5.3 Performance and Omission Faults

In our algorithm, we have implicitly assumed that all the n messages sent by M are received by S when it estimates the time on M 's clock. However this assumption may be violated if processes or communication channels have performance or omission faults which cause messages to be delayed or lost. In this section we sketch schemes to handle delayed or lost messages.

If a d_{max} does not exist, then a value for d_{max} is *chosen*. This value is chosen such that the probability of the end-to-end delay of a message exceeding this value is small. (Note that the synchronization messages sent earlier can afford to have a delay that is larger than the delay of the message that is sent last). In addition, an extra n_x synchronization messages are transmitted. The number n_x is chosen so that, the probability that more than n_x of the transmitted messages don't arrive within time, is smaller than a desired bound (η), i.e.,

$$P [Ev(n_x)] < \eta_1 \quad (0 \leq \eta \leq 1)$$

where $Ev(n_x)$ denotes the event that n_x or more messages have missed the deadline.

Alternatively, one can choose a d_{max} as before. But the slave process now resynchronizes with the master process either after it has received all the synchronization messages or after $(1 - \rho)R_{synch}$ units of time (measured on its clock) have expired, depending on which event occurs earlier. If the latter event occurs first, then the slave process will have to estimate the time on the clock of the master process using a smaller number of messages and consequently the probability of the transmission error exceeding the specified maximum transmission error will be higher.

3.5.4 Master Process Failures

We have addressed the problem of handling master process failures in [6], where we discuss two schemes to improve the fault-tolerance of the clock synchronization algorithm proposed in this chapter. The two schemes are adaptations to suit our algorithm, of schemes proposed in [22] to handle master node failures. The first

scheme is based on passive redundancy and can handle only one kind of fault, namely master node failure. The second scheme is based on active redundancy and can handle more general classes of faults. This scheme requires multiple master processes each of which is synchronized with a common external reference of time. The reader is referred to [6], for further details regarding these schemes.

3.6 Summary

In this chapter, we presented a detailed analysis of our clock synchronization algorithm. Among other things, we derived an expression for the minimum number of synchronization messages required to guarantee a specified maximum deviation at resynchronization and a specified probability of invalidity. We considered three analytical bounds on the probability of invalidity, viz., the *Tchebysheff*, *error function* and *exponential* bounds, and showed that the probability of invalidity decreases exponentially (or better) with the number of synchronization messages.

There are a number of aspects of our algorithm that can be improved through further research. Unlike most other work in this area, whose main focus is on fault-tolerance, the focus of our work has been on achieving a smaller maximum deviation between clocks in a distributed system, than previously accepted bounds. Improvement of the fault-tolerance of our algorithm, by identifying new schemes to handle various kinds of failures, is one area of research. Another area of research is to identify how the time transmission protocol proposed in this paper could be combined with the algorithms in [26], [31], [47], [48], [54], [53] and [63] to overcome the limitations imposed by Eq. (1.2). Finally, it would be interesting to identify other kinds of probabilistic algorithms, and to determine the theoretical and practical bounds on the maximum deviation between clocks in a distributed system, that can be guaranteed by probabilistic clock synchronization algorithms.

With this we conclude the presentation of the clock synchronization aspects of this research. In the following chapters, we describe our work on real-time communication.

CHAPTER 4

REAL-TIME COMMUNICATION

4.1 Introduction

We introduced the need for real-time communication in Chapter 1. *Distributed* real-time systems based on local area networks are becoming increasingly common for a number of reasons including performance advantages and fault-tolerance. Communication between tasks resident on the same or different nodes is an important activity in distributed real-time systems. Messages characterized by timing constraints will be exchanged between both application tasks and operating system tasks. Consequently, the operating system and the underlying network in a distributed real-time system must provide various facilities that take into account timing constraints of messages. This chapter deals with networking support for the communication requirements that arise in such systems.

This chapter is organized as follows. In Section 4.2, we discuss the approach to real-time communication in today's systems and illustrate the limitations of this approach and the requirements for future systems. In Section 4.3, we give a brief description of existing local area network architectures, and in Section 4.4, we propose RTLAN, a new local area network architecture for communication in distributed real-time systems. In Section 4.5, we describe the logical link control layer of this architecture. In Section 4.6, we describe the medium access control layer of RTLAN, and look at several MAC protocols that may be used at this layer to support the requirements of the LLC layer. In Section 4.7, we conclude the chapter with a brief summary.

4.2 Real-Time Communication

The term "real-time communication" may be used to describe any kind of communication activity in which the messages involved have timing constraints associated with them. For example, packet-switched voice communication, in which the individual voice packets have maximum delay constraints associated with them, is often termed real-time communication. However, in this thesis we restrict this term to mean *communication in distributed real-time systems*. While some of the protocols developed here may be applicable to voice communication, we do not consider that application any further in this paper.

Communication requirements in a distributed real-time system are induced by the need for interaction between various entities in the distributed system. Messages that arise in a distributed real-time system may be classified into two categories:

1. *Guarantee Seeking Messages*: These are messages typically critical or essential for the proper operation of the system. The requirements of these messages include a *guarantee* from the system that, if the activity that gives rise to them is accepted for execution, their timing constraints will be met with certainty.
2. *Best Effort Messages*: These are messages, typically with soft timing constraints, that do not require a *guarantee* from the system that their timing constraints will be met. However the system will try its best to satisfy the timing constraints of these messages, since minimizing the number of such messages whose timing constraints are violated will result in increased value (in some sense) for the system.

In the current generation of open loop distributed real-time systems [94], the guarantee seeking messages are of two types, viz., *periodic* messages and *alarm* (alert) messages. Periodic messages, as the name implies, are messages that are transmitted periodically. They typically carry sensor information about the current state of the controlled process. For example, in an industrial process control system, computers at various sites in the plant *periodically* collect information about the

state of the process such as flow rates, pressures and temperature, with the help of sensors, and transmit this state information to a central control room, where human controllers make decisions on the basis of this information. Periodic messages require guarantees that their delivery deadlines will be met in order to ensure that the actual state of the controlled process and the controller's view of the state obtained through these messages are close to each other. *Alarm* messages are used to disseminate information in an emergency situation. For example, in an emergency, the controller may have to shut down certain devices within a predetermined amount of time. Even though alert messages arise very rarely, due to their critical nature the system has to guarantee that alert messages will be delivered within their deadlines.

Examples of best effort messages in today's systems include some classes of commands from human controllers, and some classes of advisories and responses. For example, certain status and control messages, and trajectory advisory and response messages in the space shuttle mission control system [94] depicted in Figure 1.1 may be classified as best effort messages. These messages occur asynchronously and are usually treated as soft real-time messages. The system is designed to minimize the response time for these messages.

Most current work in real-time communication ([57],[58], [59],[94],[95]) is based on the above model of communication in distributed real-time systems, i.e., they assume that the guarantee seeking messages are either *periodic* or occur *rarely*. They also assume that the characteristics of these messages are *statically* specifiable. However this conventional model of communication requirements is likely to prove inadequate for the autonomous real-time systems of the future. These systems will be characterized by *dynamic* activities with *several* types of timing constraints [90]. *Cooperation requirements* of distributed problem solving software that will replace cooperating human controllers, and distributed real-time operating system software (e.g., distributed scheduling ([76], [75])) will necessarily induce *richer communication patterns* than periodic state message communication. The system will have to provide mechanisms to predictably handle, in addition to the traditional communication requirements, *dynamically* spawned activities with communication requirements that will include *general* kinds of timing constraints. For example,

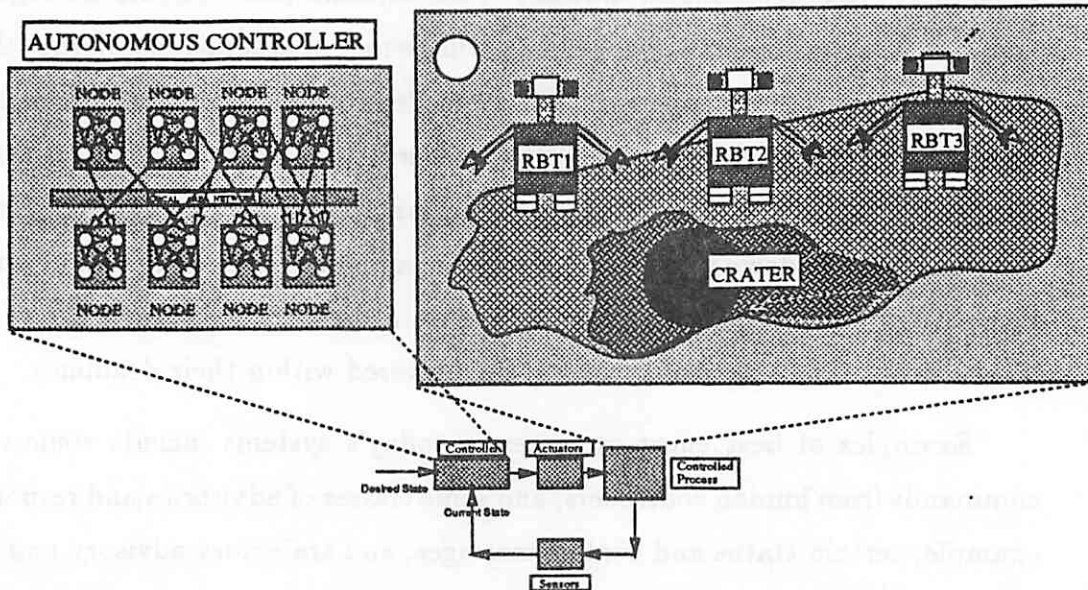


Figure 4.1 Cooperating team of robots

each message in the set of messages involved in a distributed activity may have its own individual timing constraints that are independent of other messages. Consider the scenario depicted in Figure 4.1. A team of telerobots on a planet is coordinated by an autonomous mission controller. The object of the mission is to explore the planet. If the controller detects a crater near the current location of the robots that appears to be worth exploring, then it might decide to command the robots to explore the crater, if this (dynamically arising) activity can be accommodated into the system's schedule without violating the deadlines of other scheduled critical activities. In order to ensure predictability, the controller will first try to seek a *guarantee* that the timing requirements of this cooperative activity (including the timing requirements of all the messages that arise because of cooperation) can be met before actually committing itself to exploring the crater. Many of the messages exchanged in this high level cooperative activity will be aperiodic, since this is not a low level sensing activity. Depending on how essential they are to the progress of the exploration, some of the messages exchanged will require guarantees, while others will be best effort messages. The exact details of the communication involved

in the activity, including the number of messages exchanged, their contents and individual timing constraints such as arrival times and deadlines will depend on both the activity and various dynamic factors such as the nature of the terrain near the crater, the number of robots assigned to this task and the desired degree to which the crater is to be explored.

While the above example may sound futuristic¹, it serves well to illustrate an important requirement of the evolving generation of autonomous real-time systems, viz., support for dynamic guarantees of communication activities with general types of timing constraints. The challenge in designing operating systems and communication mechanisms for the autonomous real-time systems of the future is to develop mechanisms that support these more complex requirements. The rest of this paper addresses network support for the *communication requirements* that arise in these systems.

4.3 Local Area Network Architectures

A computer network is a collection of geographically dispersed computers interconnected through a communication network. Depending on the geographic span of the network, a network may be classified² either as a local area network (LAN) or a wide area network (WAN). Distributed real-time systems are typically based on a LAN, and therefore we restrict our attention to LANs in this paper. A local area network is a network that spans a limited geographical area (0.1 km - 10 km) such as a building or a campus. A LAN is typically characterized by high speed, low error rates and ownership by a single organization.

¹In fact, considerable effort is currently being invested to realize such autonomous systems ([38], [104], [11])

²A finer classification, though not relevant to this paper, would include metropolitan area networks (MAN) that span distances of the order of 50 km and thus fall in between LANs and WANs in terms of geographical extent.

A *network architecture* defines a set of communication services, and protocols and message formats for the implementation of these services. In order to modularize and simplify implementation, modern network architectures are typically structured in terms of a set of functional *layers*. For example, the Open Systems Interconnection (OSI) reference model proposed by the International Standards Organization consists of 7 layers, viz., physical, data link, network, transport, session, presentation and the application layers. Each layer offers certain services to the immediately higher layers shielding them from the details of the implementation of these services. The services offered by a layer are implemented through a set of protocols that operate at that layer.

Two different classes of services that can be offered by the various layers of a network are *connection-oriented service* (COS) and *connectionless service* (CLS) [45]. COS is based on the establishment of a logical channel known as a connection. It is characterized by three phases, viz., *connection establishment*, *data transfer* and *connection release*. COS is typically suited for communication involving a long data transfer phase or a logically related sequence of messages, e.g., file transfer applications. Since a context is available (namely the logical connection) within which individual units of data passed between the communicating entities can be logically related, COS has smaller control overheads and can provide sequencing, flow control and error recovery. A network may use either a connectionless or connection-oriented mode of operation *internally* in order to provide communication services to its users. An *internal* connection in a wide area network that uses the connection-oriented mode of operation (e.g., TYMNET) internally is called a *virtual circuit*. For this reason, connection-oriented service is sometimes referred to as *virtual circuit service*.

The second category of communication services, connectionless service (CLS), as the name implies, is characterized by the absence of a logical connection between sender and receiver. There are no distinct phases since there is no connection to be established or released. Each unit of data is entirely self-contained and since there is no context in the form of a logical connection, the overhead information necessary to deliver the data to the receiver is duplicated in each unit of data. CLS is simpler and typically suited for short communications. However CLS does not

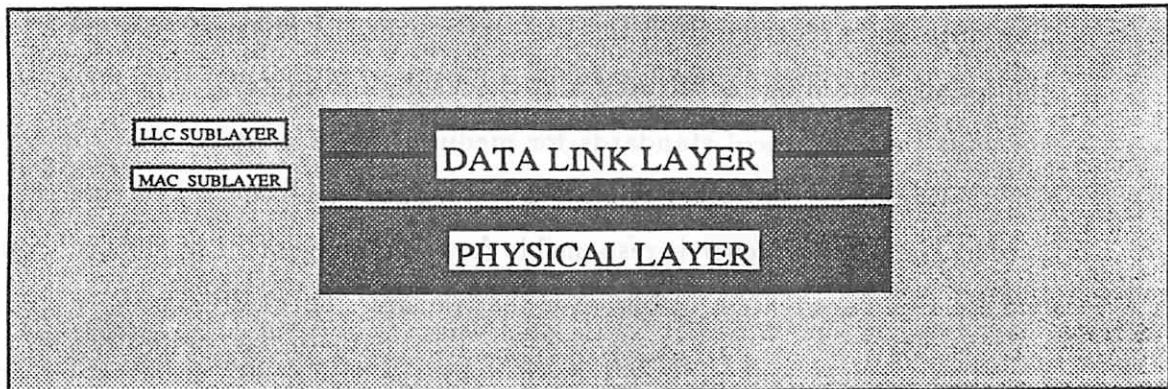


Figure 4.2 IEEE 802 LAN architecture

provide sequencing, flow control or error recovery. In networks (e.g., ARPANET) that use the connectionless mode of operation *internally*, the independent packets involved in this operation are referred to as *datagrams*. Hence connectionless service is sometimes referred to as *datagram service*.

The architecture proposed by the IEEE Project 802 committee for local area networks (Figure 4.2) may be used to illustrate these concepts. This is a simple architecture that addresses only the lowest two layers in the OSI reference model, viz., the physical and data link layers. However the data link layer provides transport layer functionality in the case of an isolated (i.e., not internetworked) LAN and provides both connectionless (or Type 1) service and connection-oriented (or Type 2) service. These two classes of service are sufficient for many applications. If an application requires higher level functionality, it must implement it itself.

The data link layer in the IEEE 802 architecture is divided into two sublayers, viz., *logical link control* (LLC) sublayer and *medium access control* (MAC) sublayer. The *LLC sublayer*, specified in the IEEE 802.2 Standard [34], is responsible for implementing medium-independent data link functions, such as connection management, error handling and flow control, and has the overall responsibility for the exchange of data between nodes. The main function of the *MAC sublayer* is the

management of access to the shared physical channel. It is responsible for transmitting data units received from the LLC layer over the physical channel after adding the required framing, addressing and checksum information. The 802 Architecture specifies three protocol standards for medium access control, viz., the CSMA/CD (802.3 standard), the token bus (802.4 standard) and the token ring (802.5 standard). The *physical layer* is responsible for the management of physical connections and for the transmission of bits over the transmission medium.

The services and protocols defined by the IEEE 802 architecture and other network architectures, although sufficient for many applications today, have an important limitation. Comer and Yavatkar [20] point out that existing protocols do not make provisions for applications to specify their performance needs such as maximum delay, minimum throughput, maximum error rate etc., and existing network architectures do not have mechanisms to meet and guarantee these performance requirements. While they make this observation in the context of research in voice and video communication in future wide area networks, a similar observation may be made in the context of distributed real-time systems. It is important for tasks executing in a distributed real-time system to be able to specify their performance requirements including timing constraints of individual messages to the operating system, and for the operating system and the underlying network to provide support for meeting and guaranteeing these requirements. However existing operating systems and networks provide little support for this. For example, the Type 1 LLC service in the IEEE 802 architecture does not take timing constraints of messages into account explicitly, and the Type 2 service does not try to guarantee timing requirements of connections. Below, we propose the elements of RTLAN, a new local area network architecture for communication in distributed real-time systems, that alleviates this deficiency.

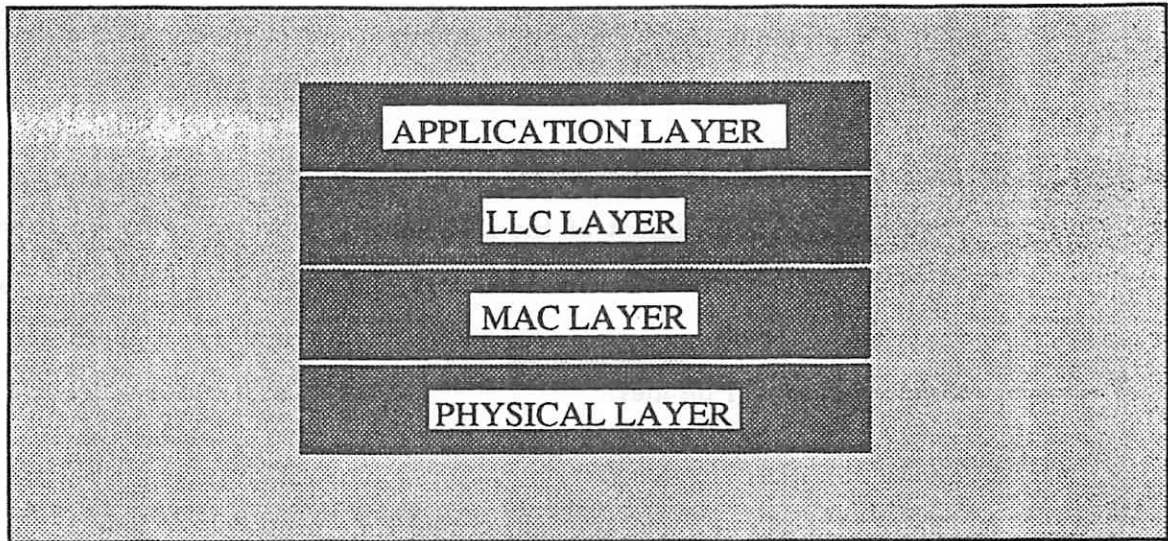


Figure 4.3 RTLAN architecture

4.4 RTLAN

The RTLAN (real-time local area network) architecture is a local network architecture for communication in distributed real-time systems, that permits applications to dynamically specify their communication timing requirements and provides mechanisms to guarantee these requirements, if needed and if at all possible. RTLAN is targeted for complex embedded control applications and so we do not consider internetworking aspects. We therefore propose a simple four layer structure for RT-LAN (Figure 4.3), along the lines of the IEEE 802 architecture. The four layers are the physical layer, the medium access control (MAC) layer, the logical link control (LLC) layer and the application layer. Some of the salient features of the RTLAN architecture are listed below:

1. *Real-Time Applications:*

RTLAN is targeted for complex real-time applications which have time-constrained communication requirements that range from simple best effort delivery requirements to dynamic guarantees of general timing requirements.

2. *Time-Constrained Services:*

RTLAN provides both connection-oriented and connectionless services, both of which consider the timing requirements of applications.

3. *LLC Layer supports Guarantee:*

Connection establishment at the LLC level is more complicated than in conventional architectures. The LLC layer incorporates *scheduling* algorithms that take a set of message timing requirements and try to *guarantee* that the requirements will be met.

4. *Real-Time MAC Protocols:*

The MAC layer employs specialized real-time protocols to help the LLC layer provide its real-time services. Some of the protocols are geared to supporting the connectionless class of service, while others are geared to supporting the connection-oriented class of service.

5. *Multiple Physical Channels:* The physical layer consists of multiple physical channels and interfaces for fault-tolerance and for meeting performance and functional requirements.

We describe the various aspects of the RTLAN architecture in more detail below, focusing mainly on the LLC and MAC layers. In order to maintain readability, wherever possible we stick to natural language in preference to OSI terminology. We also discuss only those elements of the architecture that are either novel or relevant to real-time communication. Thus we have omitted certain routine aspects such as protocol data unit structures and the details of link control rules of procedure.

4.5 RTLAN LLC Layer

The logical link control layer provides communication services to the layer above it by implementing functions that are responsible for medium-independent data link functions such as connection management, error handling, flow control and fragmentation. The RTLAN architecture distinguishes itself from a conventional

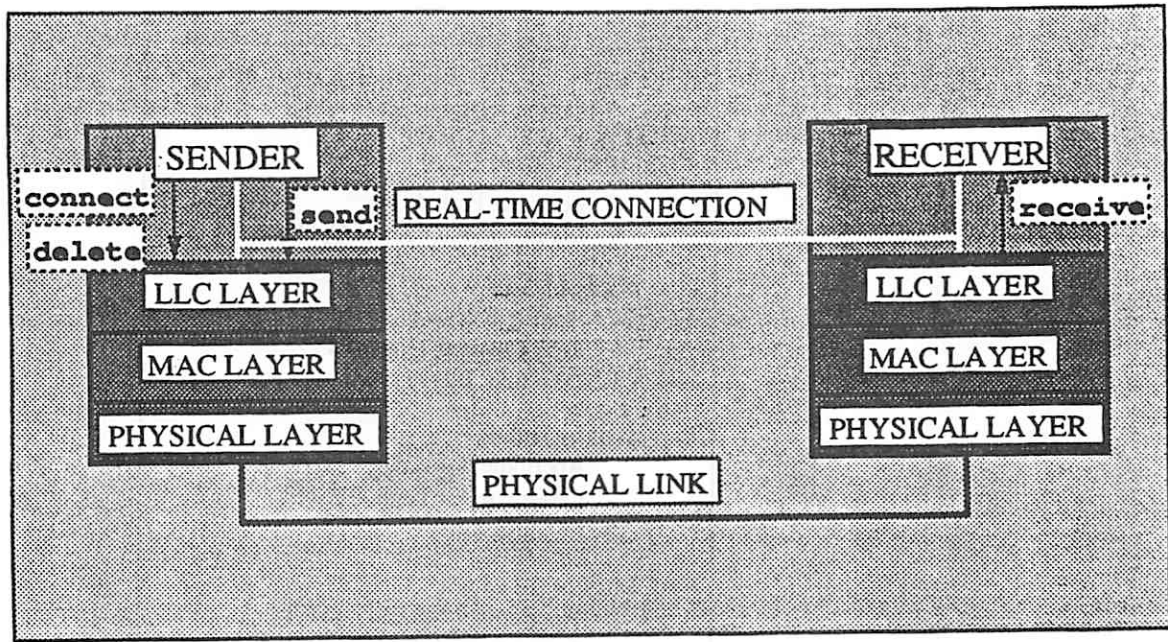


Figure 4.4 Real-time connection

LAN architecture by providing new classes of connection-oriented and connectionless services, that provide support for meeting the timing requirements of application messages. In order to meet the timing requirements of messages, it also takes an unconventional approach to error handling and flow control. We describe these aspects of the LLC layer in the following sections.

4.5.1 Services

The LLC layer in RTLAN offers a connection-oriented service known as RTCOS (real-time connection-oriented service) and a connectionless service known as RTCLS (real-time connectionless service). These services are accessible to applications through LLC service access points.

4.5.1.1 Real-Time Connection-Oriented Service

RTCOS is a connection-oriented service that permits the sender to specify its timing requirements at the time of connection establishment. RTCOS is meant for supporting the requirements of the class of guarantee-seeking messages. The service is characterized by the establishment of a logical connection known as a *real-time*

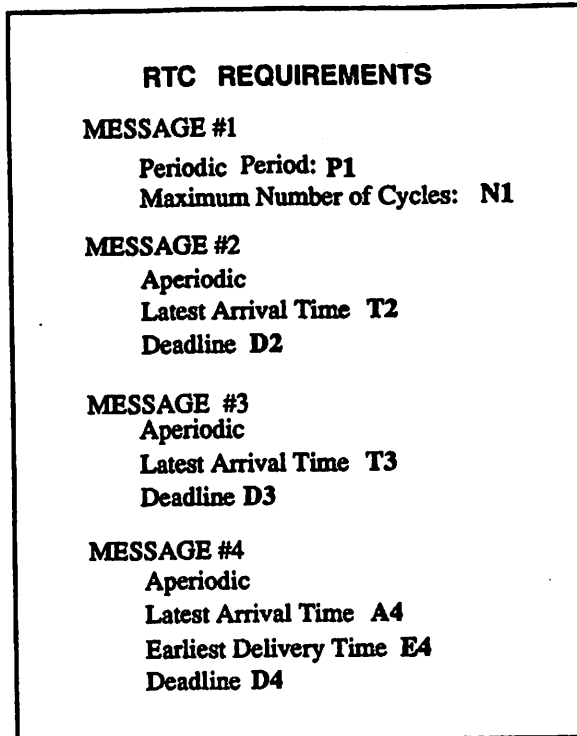


Figure 4.5 Real-Time connection requirements

connection. A real-time connection (Figure 4.4) represents a simplex end-to-end communication channel between two communicating application level entities, a *sender* and a *receiver*. In order to set up a real-time connection, the sender specifies the timing requirements of the messages that it plans to send over the connection to the LLC layer at the time of connection establishment (connection establishment is done at the time of scheduling a task; the connection request is typically made by the operating system, which is also part of the application layer, on behalf of the sender). The timing constraints may be fairly general and may include periodicity, arrival time, laxity, deadline, etc. Figure 4.5 depicts an example of the requirements that may be specified by an application task to the LLC layer. In this example, the application task requires guarantees that the timing constraints of a session involving four messages will be satisfied. The first message is periodic and a guarantee is requested for N1 consecutive instances of the message. The remaining three messages are aperiodic with various arrival time and deadline requirements. As in a typical connection-oriented service, the service provider tries to set up the requested connection through a process of negotiation that may involve the sender and the

receiver in addition to itself [45]. The connection is set up only if the specified requirements can be guaranteed; otherwise the sender is informed that the connection cannot be established. In order to set up the connection, the RTCOS service provider employs the services provided by the MAC layer and suitable scheduling algorithms, which we discuss later. The following operations that comprise real-time connection-oriented service summarize the above descriptions:

- **rtcid** ← **connect(receiverid, requirements)**

The communication service provider checks to see if it can set up a connection that satisfies the specified real-time requirements. If so, it returns a real-time connection identifier; otherwise it returns an error code.

- **send(rtcid, message)**

Sender requests delivery of a message to the receiving end of the specified real-time connection.

- **message** ← **receive(rtcid)**

Receiver requests receipt of a message sent over the specified real-time connection.

- **delete(rtcid)**

Sender or receiver requests termination of specified real-time connection.

4.5.1.2 Real-Time Connectionless Service

RTCLS is an unreliable *connectionless* service used for transmitting time-constrained messages. It is unreliable in the sense that the timing constraints of messages transmitted using this service may not be satisfied. However RTCLS tries to deliver messages within their timing constraints on a *best effort* basis. Thus this service is suitable for the class of best effort messages. By best effort, we mean that at each decision making point within the service, decisions are made on the basis of timing constraints of the pending packets. For example, if there are several packets waiting to be transmitted, then the system would try to transmit them in an order

that minimizes the number of messages whose deadlines are not met. Since real-time connectionless service does not involve setting up a connection, it is defined by a simpler set of operations:

- **send(receiver, message, requirements)**

Sender requests delivery of a message to the specified receiver; sender also specifies timing requirements (e.g., a deadline for the message) that it would like to be met if possible.

- **(message, sender) ← receive()**

Receiver requests receipt of a message.

In order to support RTCLS, the LLC layer makes use of the services provided by suitable protocols at the MAC layer that explicitly consider timing constraints of packets in arbitrating access to the medium.

4.5.2 Fragmentation

One of the functions implemented by the logical link control layer is the transformation of messages provided to it by the application layer to a form suitable for the medium access control layer. One step involved in this function is known as *fragmentation* or *packetization*. This refers to the division of a long message into smaller packets or *frames* that satisfy the maximum packet length requirements of the medium access control layer. Since messages have timing requirements associated with them, the LLC layer propagates these requirements to the individual packets, by propagating the requirements of a message to each of its fragments. For example, the deadline of a message could be copied to all its fragments, or the deadlines of the fragments can be staggered such that the last fragment is assigned the deadline of the message and the leading fragments are assigned earlier deadlines. We refer to a fragment that is derived from a RTCLS message as a *real-time datagram*, in analogy with datagrams in a wide area network.

4.5.3 Guaranteeing Real-Time Connections

One of the distinguishing characteristics of the RTLAN architecture is its connection-oriented communication service, RTCOS, that permits the sender to specify its timing requirements at the time of setting up a connection and seek a guarantee from the system that these timing requirements will be met. In this section, we examine mechanisms that the LLC layer may use in order to provide such a guarantee.

4.5.3.1 Priority Assignment Approach

The first approach that we discuss assumes that all the messages are periodic and statically specified. By dedicating a separate physical channel for these messages, this approach may be used to handle the static periodic message components of systems that involve both static and dynamic communication requirements.

This approach is based on the *rate monotonic* priority assignment scheme [62]. The rate monotonic priority assignment scheme, originally developed in the context of scheduling periodic tasks on a uniprocessor, is a fixed priority assignment scheme in which tasks with a smaller period (i.e., higher rate) are assigned higher priorities. Scheduling then consists of merely allocating the processor to the pending task with the highest priority, preempting the currently running task if necessary. Liu and Layland [62] have shown that the rate monotonic priority assignment scheme is optimal in the following sense - if some priority assignment scheme can assign suitable priorities to tasks such that every task will complete within its period, then the rate monotonic priority assignment scheme can do so. They also show that a sufficient condition for such a priority assignment to exist is that the sum of the utilizations of the individual tasks must satisfy

$$\sum_{i=1}^n U_i \leq n \left(2^{\frac{1}{n}} - 1 \right), \quad (4.1)$$

where n is the number of tasks and U_i is the utilization of task i defined as

$$U_i = \frac{C_i}{T_i},$$

where C_i and T_i are respectively the computation time and period of the task. Lehoczky and Sha [57] have extended this result to the problem of scheduling n periodic messages on a shared bus. Strosnider [94] has further extended this result to the *deferrable server* algorithm that makes use of a periodic server to service aperiodic messages. He has used this algorithm to provide guarantees for both periodic messages and a limited class of alert messages that are assumed to occur rarely.

Guaranteeing an application's message timing requirements, in the priority assignment approach, consists of merely assigning fixed priorities to each of the periodic messages (and the periodic server that services aperiodic messages) involved (on the basis of their periods), and ensuring that the utilizations of the messages satisfy Eq. (4.1). Actual implementation of priority arbitration is left to suitable MAC protocols.

4.5.3.2 Real-Time Virtual Circuit Approach

An alternative approach, that may be used to guarantee the timing requirements of both statically known and dynamically arising messages, is to use the notion of *real-time virtual circuits* (RTVCs). An RTVC is a *logical channel* that has the property that the *service time* of a packet queued on this channel, the length of the interval between the instant at which the packet enters service and the instant at which transmission of the packet completes successfully, is bounded for a fixed packet length. Thus the LLC layer can assume that once a packet queued onto a RTVC has been accepted for service, it will be transmitted within a bounded amount of time. This bound is determined by the MAC protocols used to implement RTVCs.

The LLC layer makes use of RTVCs as follows. Each RTVC has a transmission queue associated with it. When an application entity requests a real-time connection

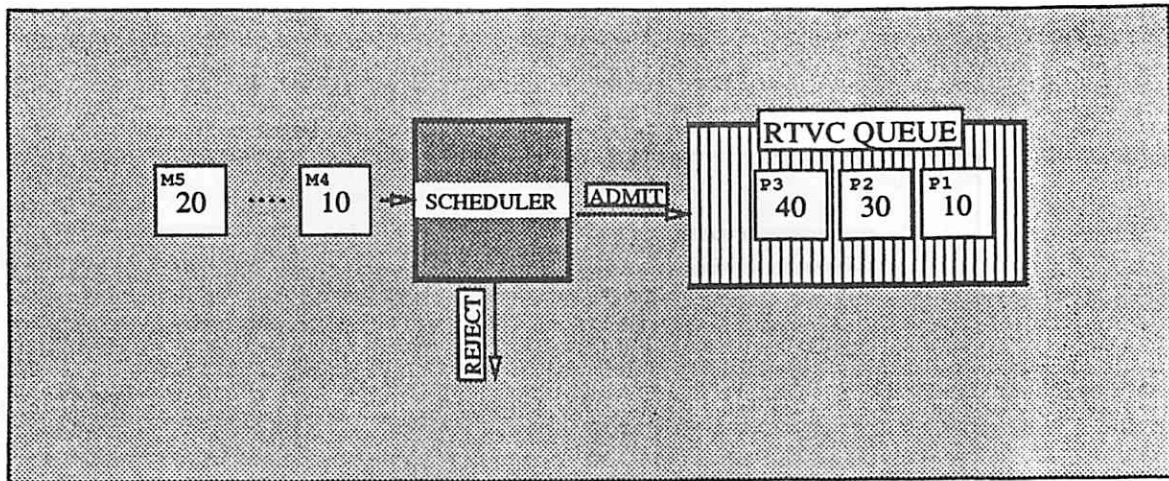


Figure 4.6 Real-time virtual circuit scheduling

from the LLC layer, the LLC layer firsts *fragments* messages in the request that are longer than the maximum packet length into multiple packets and propagates the timing constraints of messages to their fragments. The set of fragments are then passed on to a LLC layer entity known as the *scheduler*. The scheduler takes a set of message fragments with timing requirements, and applies a *scheduling algorithm* [18] to determine if the set of fragments can be inserted (according to some insertion discipline, such as the first-in first-out (FIFO) or the minimum laxity first (MLF) discipline³) into the queue associated with some RTVC, without violating the timing constraints of the packets that have already been admitted into the queue and that are awaiting transmission. In order to make this determination the scheduler makes use of a worst case assumption, since the scheduler cannot predict the service time exactly. The assumption made is that each packet in the queue will have a service time equal to the worst case service time. Such a worst case service time is guaranteed to exist, since an RTVC by definition has a bounded packet service time. The example shown in Figure 4.6 illustrates these ideas. In this example, there is one RTVC (with a worst case service time of 10) which already

³A commonly used discipline in the scheduling of real-time tasks is the *minimum lazity first* (MLF) discipline, which is known to be optimal in the following sense - *if some discipline can schedule a set of independent tasks so that all the tasks meet their deadlines, then the minimum lazity first policy can do so.*

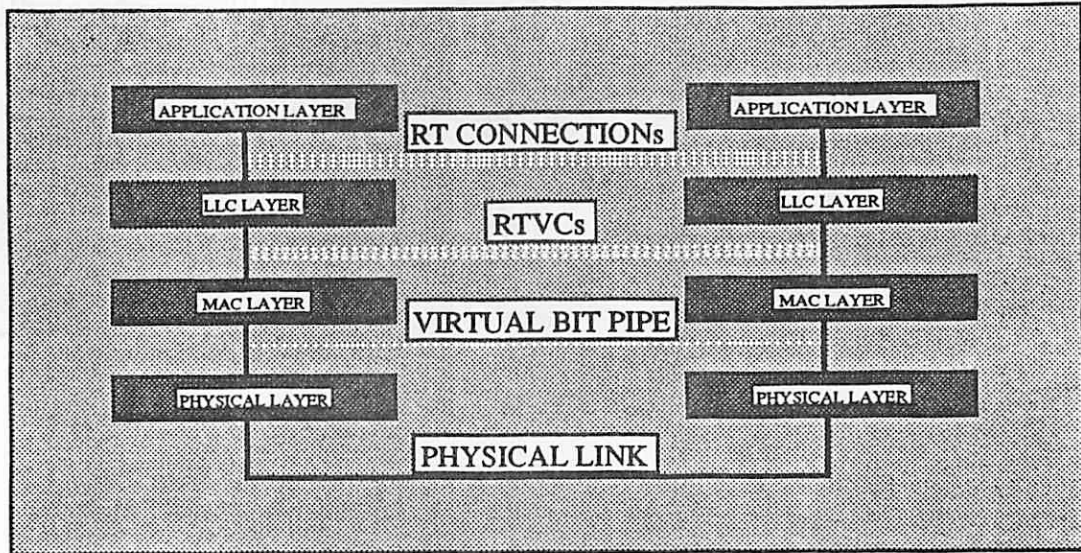


Figure 4.7 Abstractions provided by RTLAN layers

has three guaranteed packets waiting to be transmitted. Packet M1 has a laxity (time until deadline for start of transmission) of 10, M2 has a laxity of 30 and M3 has a laxity of 40. If a packet M4 of laxity 10 arrives, then it cannot be admitted into the system, if the MLF discipline is used, since the deadlines of both M1 and M4 cannot be simultaneously met. However a packet M5 with a laxity of 20 can be admitted, since it is possible to meet its laxity requirements without violating the requirements of any of the messages already in the queue. It should be pointed out that, for the real-time virtual circuit approach to guarantee a reasonable fraction of the connection requests that are made, the worst case packet service time must be of the same magnitude or smaller than the average laxity of the packets involved.

Note that RTVCs are abstractions provided by the MAC layer to the LLC layer, Figure 4.7 illustrates how each layer in the architecture provides an abstraction that is used by the layer at the immediately higher level to implement its services. The physical layer receives a bit stream from the MAC layer and converts the bits into *electrical signals* that are transmitted over the physical link. The physical layer

hides the physical details of the link and thus provides the abstraction of a virtual *bit pipe* [12] to the MAC layer entities. Protocols in the MAC layer make use of this raw *multiple access* bit pipe with *potentially unbounded packet service times* to provide the abstraction of logical channels with bounded packet service times, viz., RTVCs. The LLC layer employs these channels to provide the abstraction of a real-time connection to the application layer.

Note that the MAC layer has to employ suitable protocols in order to provide such an abstraction. For instance, an Ethernet based system is unsuitable, since a message that has entered service may *never* get transmitted because of the randomized collision *resolution* strategy used in the Ethernet MAC protocol. In Section 4.6.1.2, we will look at several medium access control protocols that may be used to realize the RTVC abstraction.

4.5.4 Flow Control and Error Control

Flow control and error control are two important functions for which the LLC layer is responsible. Flow control is a synchronization technique to ensure that a sending entity does not overwhelm a receiving entity with data, causing its buffers to overflow. Flow control is typically implemented through acknowledgement based sliding window protocols. Error control mechanisms are responsible for detecting and correcting errors that occur in the transmission of packets. These mechanisms are required because it is possible for a transmitted packet to be *lost* or *damaged* because of noise on the communication channel. Error control is typically implemented through ARQ (automatic repeat request) mechanisms. In an ARQ mechanism, the receiver sends a positive acknowledgement to the sender, if it receives an undamaged packet (damage is typically detected through the use of an error-detecting code such as cyclic redundancy check); if it receives a damaged packet, it sends a negative acknowledgement. If the sender receives no acknowledgement within a timeout period or receives a negative acknowledgement, then it retransmits the packet to the receiver.

In the real-time connection oriented service, flow control schemes based on sliding window protocols where the receiver may block the sender by withholding acknowledgements, are inappropriate. If a receiver can block the sender for an arbitrary amount of time (because of an insufficient number of buffers), then the timing constraints of guaranteed packets may be violated. In order to guarantee the timing constraints of packets, the required number of buffers will have to be *reserved* at the destination node. This will permit the sender to transmit its packets whenever they are ready, rather than wait for permission from the receiver. This *buffer reservation* is part of the three way negotiation involved in setting up a real-time connection. When a connection is requested under any of the approaches described in Section 4.5.3, the LLC layer makes sure that sufficient buffer space is reserved at the destination nodes for the packets involved in the request. In a real-time connectionless service, flow control can be exercised by merely dropping packets that arrive when the receiver's buffers are full. This is acceptable, since RTCLS does not offer any guarantee on the delivery of packets.

Two kinds of error control schemes may be used for RTCOS. The first scheme is an ARQ scheme based on *temporal redundancy* that is appropriate when the deadlines of the packets involved in a connection are large enough to permit the use of timeouts. At the time of connection establishment, the maximum number of retransmissions possible for each packet is computed based on the packet's deadline and the retransmission timeout (alternatively, this number may be included in the specification of the requirements for the connection). The scheduler takes into account all the potential retransmissions in determining whether the requested connection can be guaranteed or not. When the packet is actually transmitted, the receiver returns an acknowledgement (transmitted as a real-time datagram) to the sender. If the acknowledgement is received within the retransmission timeout period, then all the remaining duplicates are dequeued; otherwise, the next duplicate is transmitted, when its turn comes. A variation on this scheme is to use temporal redundancy and avoid acknowledgements all together [46]. In this variation all the duplicates are transmitted, irrespective of whether or not the earlier duplicates

suffer transmission errors, leaving it to the receiver to select an error-free duplicate and discard the others.

The second scheme is based on *spatial redundancy* and is appropriate when the deadlines of the packets involved in a connection are too small to permit re-transmissions based on timeouts. In this scheme, the LLC layer tries to schedule a connection request on multiple physical channels. A request for a real-time connection is guaranteed only if it is possible to redundantly schedule the request on the specified number of physical channels. Thus multiple copies of each message involved in an accepted connection are transmitted on multiple channels, thereby improving the probability of error-free receipt. The receiver is then responsible for discarding duplicates.

Note that neither of the above schemes can guarantee reliability with certainty; they can only provide a conditional guarantee that if the number of faults (errors) does not exceed a certain number, then a packet will be transmitted successfully because of the redundancy in the system. However this is true of *any* mechanism for fault-tolerance. RTCLS does not require any error control mechanism, since it does not guarantee reliability.

4.6 RTLAN MAC Layer

A local area network is typically based on a shared physical channel such as a bus or a ring, commonly referred to as a multiple access channel. Since multiple nodes may simultaneously contend for access to this shared channel, a mechanism is needed to manage access to it. The main function of the medium access control layer is to arbitrate access to the channel. The MAC layer implements this arbitration mechanism through a suitable *multiple access protocol*. Kurose, Schwartz and Yemini [50] provide a good survey of multiple access protocols. Multiple access protocols can be classified into *real-time* and *non-real-time* protocols on the basis of whether or not they provide support for real-time communication. Many protocols that have been proposed in the literature, such as the Aloha protocol [1], the CSMA protocols in [43] and the Ethernet protocol [69] are non-real-time protocols since

they do not incorporate any notion of packet timing constraints. In this section, we look at examples of real-time protocols that have been proposed in the literature that may be used to provide support for RTCOS and RTCLS.

4.6.1 RTCOS/MAC Protocols

In Section 4.5.1.1, we defined RTCOS as a connection-oriented service that permits an application layer entity to specify its communication timing requirements to the LLC layer and seek a guarantee that these requirements will be satisfied. We also described three different approaches to providing such guarantees, viz., *priority assignment* approach, *real-time virtual circuit* approach and *reservation* approach. The LLC layer invokes the services of the MAC layer in order to provide such guarantees. In this section, we look at several MAC protocols that may be used to support the services required by the LLC layer, for each of these approaches. The priority assignment approach requires priority resolution protocols at the MAC layer; the real-time virtual circuit approach requires MAC protocols that can guarantee bounded packet service times and the reservation approach requires reservation protocols.

4.6.1.1 Priority Resolution Protocols

In Section 4.5.3.1, we described a priority-based scheduling approach based on the rate monotonic algorithm and the extensions to this algorithm proposed by Lehoczky and Sha [57], and Strosnider [94]. This approach may be used to guarantee certain classes of statically specifiable messages. In order to ensure that packets are actually transmitted according to the priorities assigned to them by the LLC layers, the MAC layers will have to employ an appropriate *priority resolution protocol*. A priority resolution MAC protocol always selects for transmission, the packet with the highest priority among all the contending packets in the system.

The IEEE 802.5 standard for token ring local area networks includes a priority resolution scheme. In a token-passing ring network, access to the ring is arbitrated

through a short control packet known as a token. At any moment, only the node that is in possession of the token is permitted to transmit. A node that has completed transmission or that has no packets to transmit passes on the token to the next node. This simple token-passing scheme may be augmented to support priority resolution by including additional fields in the token. In the IEEE 802.5 standard, priority resolution is supported using two 3 bit fields in the token known as the *priority* field and the *reservation* field. The priority resolution scheme works as follows. When a station captures the token, it sets a one bit field in the token known as the *token bit* to indicate that the token has been *claimed*. It then transmits the claimed token followed by the data packet. Each node examines the claimed token as it passes by. It overwrites the reservation field in the token with the priority of its pending packet, if this priority is greater than the value in the reservation field. Thus the reservation field contains the priority of the packet with the highest priority among all the packets at the heads of MAC transmission queues at the various nodes in the system. After the transmitting station has received the token and the data packet back, it removes the data packet from the ring and clears the token bit in the token. It also copies the reservation field of the token to the priority field before releasing the free token. Any node with a pending packet with priority greater than or equal to the priority field of the free token may now capture the token. Strosnider and Marchok discuss the use of the token ring priority resolution scheme for scheduling statically specified message sets in more detail in [95].

Priority resolution protocols for bus-based systems are classified by Valadier and Powell [100] into three categories, on the basis of the underlying scheme involved:

1. *Deference delays*: In this scheme at the end of a packet transmission, each node defers transmission of its packet (if any) by a period of time whose length in round trip propagation delay units is equal to $p_{max} - p$. Here p is the priority associated with a node and p_{max} is the largest priority value possible. At the end of its deference period, each node senses the channel. If the channel is sensed to be idle, it means that no other node has a value of p greater than that of the node that sensed the channel. So this node is allowed

to transmit its packet. Thus this scheme selects the node with the largest value of p (highest priority) that has a packet to transmit. Note that the length of the deference delay, the worst case overhead (in the form of wasted channel time) per packet that arises because of priority resolution, lies in the range $[0, p_{max} - p_{min}]$. The average overhead, assuming all possible values of the overhead are equally likely, is given by $\frac{1}{2} (p_{max} - p_{min})$, i.e., it is a linearly increasing function of the number of priority levels. The protocols proposed in [29] and [97] are examples of this class.

2. *Preamble lengths*: In this scheme, at the end of a packet transmission, each node transmits a preamble signal for a period of time whose length in round trip propagation delay units is equal to $p - p_{min}$, where p_{min} is the smallest priority value possible. Thus a node with a higher priority will transmit a longer preamble signal. Collision detection is suppressed at a node during the period that its preamble is being transmitted. If a node detects a collision at the end of the transmission of its preamble, it means that there is another node with a larger value of p that is still transmitting its preamble. Only the node which does not detect a collision at the end of its preamble, i.e., the node with the longest preamble length or the largest value of p , is allowed to transmit its packet. The per packet overhead that arises because of priority resolution again lies in the range $[0, p_{max} - p_{min}]$ units of time. Thus the average overhead is again a linearly increasing function of the number of priority levels. The protocol proposed in [37] is an example of a protocol that employs this scheme.
3. *Forcing Headers*: The forcing header scheme makes use of the inherent wired-OR property of a broadcast bus, namely the property that the value of the bit received by a node from the bus is equal to the logical OR of the bits transmitted by all the nodes. In this scheme, each packet transmission is preceded by a header epoch, during which all the nodes with packets to transmit, simultaneously transmit their priorities from the most significant to the least significant bit in successive slots (of length equal to a round trip propagation delay). The nodes sense the medium as they transmit their priority bits. If

a node has transmitted a 0 bit in a slot and receives a 1 bit, it means that some node with a higher priority value is contending for access to the channel. In this case the node that transmitted the 0 bit drops out of contention. At the end of the header epoch, only the node with the highest priority value remains and this node transmits its packet to completion. The length of the header epoch is equal to the number of bits required to represent the range of values $[p_{min}, p_{max}]$, i.e., $\lceil \log_2(p_{max} - p_{min} + 1) \rceil$ time units. This represents the priority resolution overhead and is constant for all packets and hence also represents the average overhead. Thus the average overhead for this scheme increases *logarithmically* with the number of priority levels. The MLMA or multilevel multiaccess protocol [81] employs such a scheme.

It is clear that, among the above schemes for bus-based systems, the scheme based on forcing headers incurs the least overhead, because of the logarithmic relationship between the overhead and the number of priority levels in this scheme. In fact, such a scheme is used in the Distributed Systems Data Bus (DSDB) developed by IBM, which forms the hub of a large distributed real-time system [94]. In Section 5.3, we describe a priority resolution protocol based on a window splitting paradigm whose overheads are about the same as the forcing headers scheme.

4.6.1.2 Real-Time Virtual Circuit Protocols

A real-time virtual circuit is a logical channel that has the property that the service time of a packet queued on the channel is bounded. The packet service time consists of two components, the physical *channel access time* and the packet *transmission time* (Figure 4.8). The channel access time arises because the underlying physical channel in a local area network is typically a multiple access channel that is shared by all the nodes on the network; a node may therefore have to wait for the channel arbitration mechanism to permit it to transmit. The packet transmission time is the time required to transmit a packet. Note that the packet transmission time may be bounded by restricting the maximum size of a packet. However, unless a suitable medium access control (MAC) protocol is used, the channel access time

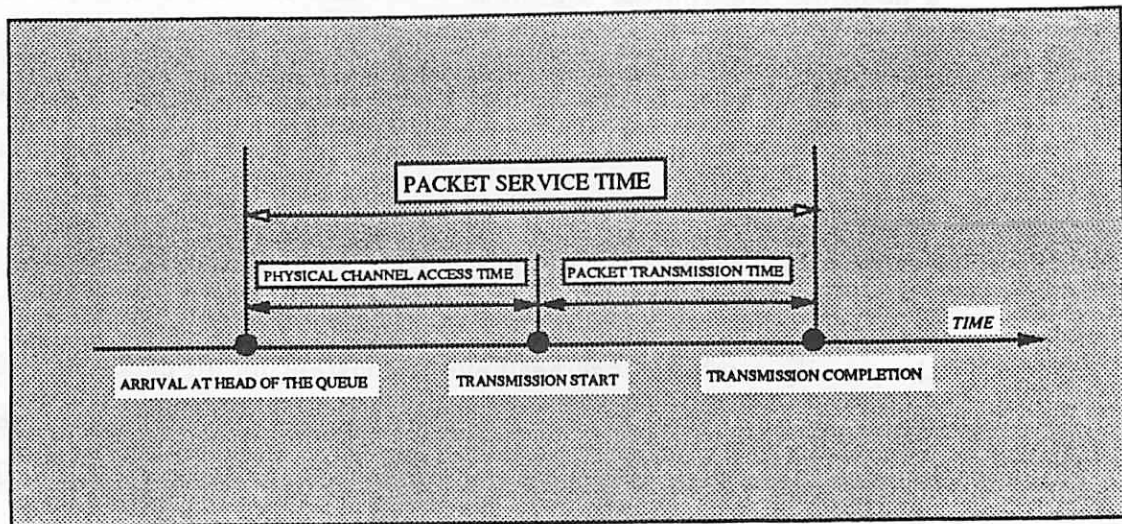


Figure 4.8 Packet service time

can be unbounded. For example, in a CSMA/CD (Ethernet) based system, a message that has entered service may *never* get transmitted because of the randomized collision *resolution* strategy used in the MAC protocol. In this section, we look at several MAC protocols, each of which can guarantee a bounded channel access time.

In Time Division Multiple Access (TDMA), time is divided into fixed length intervals known as *frames*. Each frame is further subdivided into slots with at least one slot per node. If there are S slots in a frame, then the maximum separation between two instances of the same slot is given by the frame length SP , where P is the packet transmission time (assuming that only one packet is permitted per slot). Real-time virtual circuits may be implemented by dedicating one slot per real-time virtual circuit. In this case the worst case channel access time for a packet queued on a real-time virtual circuit is given by SP .

In token-passing protocols, the nodes are organized in a logical (e.g., token bus [33]) or physical ring (e.g., token ring [36]). A real (e.g., token ring, token bus) or virtual (e.g., BRAM [19], MSAP [44]) token that circulates around this ring is used to arbitrate access to the channel. The token confers upon its holder the privilege to transmit on the channel for a bounded amount of time (the token holding time). If there are no packets to transmit, or if the token holding time has been exceeded,

the token is passed onto the next node in the ring. If the token holding time per node is P time units (i.e., a node is permitted to transmit only one packet in each cycle), then the maximum length of the interval between successive channel accesses by a node is given by $N(P + \tau)$, where N is the number of nodes in the system and τ is the token passing overhead.

The 802.3D protocol proposed by Le Lann [58] is another MAC protocol that may be used to guarantee a bounded channel access time. The '802.3' in the name refers to the IEEE 802.3 MAC layer standard [35] for CSMA/CD (the access control method used in Ethernet) which, owing to the collision resolution strategy (binary exponential backoff) used, cannot guarantee an upper bound on the channel access time. The 'D' refers to the fact that the 802.3D protocol is deterministic, i.e., it can guarantee an upper bound on the channel access time. This protocol is essentially an adaptive tree walk protocol. Normally, the nodes are in the *random access* mode in which they access the channel randomly without any coordination. Thus under low load conditions, channel access is immediate. However, the moment a collision occurs, the nodes switch to the *epoch* mode in which a tree search process is used to resolve the collision within a bounded amount of time. The worst case channel access time for the 802.3D protocol is the worst case time required to resolve a collision involving all the nodes and is equal to $N(P + \tau)$, where P is the packet transmission time and τ is the round trip propagation delay.

The notion of waiting room priorities ([100], [74]) has been used to implement the *waiting room protocol* which is characterized by bounded channel access times. The waiting room protocol is based on a logical waiting room which a packet has to enter before it can be transmitted. A packet can enter a waiting room only when the waiting room is empty. However, since packets belonging to different nodes can simultaneously find the waiting room to be empty, more than one packet can enter it simultaneously. The packets in the waiting room are then transmitted in some prespecified order (e.g., descending order of node addresses). If there are N nodes in the system and the transmission order is the descending order of node addresses, then the worst case channel access time occurs for a packet belonging to the node with the smallest address and is equal to $2(N - 1)P$ units of time [100].

Table 4.1 Minimum real-time virtual circuit packet laxities (4 Mbps channel)

Packet size	Number of real-time virtual circuits				
	10	20	30	40	50
400 bits	1 msec	2 msec	3 msec	4 msec	5 msec
800 bits	2 msec	4 msec	6 msec	8 msec	10 msec
1600 bits	4 msec	8 msec	12 msec	16 msec	20 msec

Thus a variety of approaches may be used to realize the abstraction of a real-time virtual circuit, all with approximately the same worst case channel access time (except the waiting room protocol that has a larger worst case channel access time) of approximately $N.P$. The worst case channel access time determines the minimum laxity that a packet must have in order to get guaranteed. Table 4.1 provides an idea of the magnitude of the minimum laxity that a packet must have in order to be guaranteed. The table assumes a 4 Mbits/s channel. In Section 5.5, we describe another protocol, this time based on a window splitting paradigm, that has approximately the same bound on the channel access time as most of the protocols above, but which also provides structural homogeneity.

4.6.2 RTCLS/MAC Protocols

In Section 4.5.1.2, we defined RTCLS as an unreliable connectionless service that tries to deliver messages within their timing constraints on a best effort basis. In order to provide such a service to the application layer, the LLC layer invokes suitable MAC layer services that are implemented through a class of MAC protocols which we refer to as *best effort* protocols. Best effort protocols take into account the timing constraints of the individual contending packets in arbitrating access to the shared channel. Even though these protocols do not provide the ability to guarantee that the timing constraints of messages will be satisfied, they try to minimize the number of messages that are *lost*, i.e., that do not meet their deadlines.

The window protocol proposed by Kurose, Schwartz and Yemini in [51] is an example of a protocol that considers timing constraints of messages in arbitrating

channel access. The protocol assumes that all packets have identical laxities, as would be the case in a voice communication application. The protocol effectively implements a global first-in first-out (FIFO) policy of message transmission augmented with an additional policy element that dictates the discarding of messages that have missed their deadlines before transmission.

The FIFO policy is appropriate when all the packets have identical laxities. This policy chooses the packet that arrived first, i.e., the packet that has waited longest and hence is likely to miss its deadline first (the most "urgent" packet). However other policies have also been proposed in the literature. For example, Kallmes, Towsley and Cassandras [40] have considered the LIFO (Last In First Out) policy, and shown that under certain conditions (when the deadlines are i.i.d. with concave CDFs) this policy is optimal. In [73], the minimum laxity first policy (also known as the shortest time to extinction policy) or its variations have been shown to maximize the fraction of the number of customers in a queueing system that meet their deadline under fairly general conditions. The virtual time CSMA protocol [107] and the MLF window protocol [108] try to implement a global minimum laxity first policy for message transmission on a bus-based system, i.e., they select the message with the smallest laxity in the *entire* system for transmission.

In the virtual time CSMA protocol [107], each node maintains two clocks, a real time clock and a virtual time clock that runs at a higher rate than the real-time clock. Whenever a node senses the channel to be idle, it resets and restarts its virtual-time clock. When the reading on the virtual clock is equal to some parameter value of a message waiting to be transmitted, the node transmits the message. Collisions are resolved through a random backoff procedure. Different transmission policies are implemented by using different message parameters to control the operation of the virtual clock. For example, choice of message arrival time as the parameter used by the protocol corresponds to the first-in first-out transmission policy [70], while use of message laxity corresponds to the minimum laxity first policy.

The MLF window protocol belongs to the class of inference-seeking protocols [108]. In the window protocol, a window that slides along the real-time axis is used

to identify the node that has the message with the most urgent transmission requirements (the message whose 'latest time to send' is smallest) and this node is granted transmission rights on the channel. The protocol effectively tries to implement a global minimum laxity first policy. In this protocol, each node maintains a window that spans a segment of the real-time axis. When a node has a message to transmit, it waits for the channel to become idle and then transmits the message, if the latest time to send the message falls within the current window. If there is a collision (which can occur if another node also had a message that fell within the window), then all the nodes split the window into two halves and examine the left window first. If there is only one node with a message in the left window, then that node transmits its message successfully. If there are two or more nodes in the window, then a collision occurs again and the splitting process is repeated until a message is successfully transmitted. If there are no nodes with a message whose latest time to send falls within the left window, the window is expanded to include the left half of the right window and the window examination process is repeated again. This process is continued until a message is transmitted, or the whole of the initial window has been examined. The window protocol has been shown to very closely approximate an ideal protocol that implements, without incurring any arbitration overheads, the minimum laxity first policy for message transmission [108]. An important advantage of this protocol over traditional window protocols is that a newly arriving message need not wait for all the messages currently involved in contention resolution to be transmitted, before being considered for transmission.

4.7 Summary

Most current work in real-time communication deals with static systems in which messages with timing constraints are mainly periodic, or occur only rarely. However the dynamic closed loop distributed real-time systems of the future will be characterized by richer communication patterns. Specialized network services and protocols will be required to support the communication requirements of these systems. In this paper, we proposed RTLAN, a new local area network architecture

for communication in such systems. RTLAN provides new classes of connection-oriented and connectionless services known as RTCOS and RTCLS respectively, that take the timing constraints of messages explicitly into account. In order to provide these services, RTLAN employs specialized real-time medium access control protocols. We presented several medium access control protocols that can be used at the MAC layer of RTLAN.

Medium access control protocols for bus-based systems is the focus of the next two chapters. Existing standards for medium access control protocols for bus-based systems, viz., the IEEE 802.3 and 802.4 standards, have certain limitations (discussed in the next chapter) in their support for real-time communication. We propose, develop, analyze and evaluate a new suite of real-time medium access control protocols that is devoid of these limitations, in the next two chapters.

CHAPTER 5

WINDOW MAC PROTOCOLS

5.1 Introduction

In Chapter 4, we examined the limitations of existing approaches to real-time communication, identified the communication requirements (*viz.*, guaranteed and best-effort support for timing constraints of messages) of the evolving next generation of distributed real-time systems, and proposed RTLAN, a four layered local area network architecture that supports these requirements. RTLAN provides new classes of connection-oriented and connectionless services known as real-time connection-oriented service (RTCOS) and real-time connectionless service (RTCLS), which may be used to support guarantee-seeking and best effort messages respectively. Implementation of RTCOS requires support at the medium access control layer in the form of *priority resolution* protocols that implement global packet-level priority resolution, and protocols that can *guarantee bounded channel access times*. Implementation of RTCLS requires medium access control protocols that *explicitly consider the timing requirements of messages* such as message deadlines, and arbitrate access to the channel on the basis of these requirements. In this chapter, we have considered a distributed real-time system based on a local area network with a bus topology, and addressed the problem of network support (specifically, support at the *medium access control* layer) for providing the required communication services. We propose, develop, analyze, and evaluate a new suite of real-time medium access control protocols that is devoid of deficiencies (in supporting real-time communication) found in existing standards for medium access control protocols for bus-based systems, *viz.*, the IEEE 802.3 and 802.4 standards.

The IEEE Project 802 Committee has developed two sets of medium access control layer standards for local area networks based on a *bus* topology, *viz.*, the

802.3 standard [35] and the 802.4 standard [33]. The IEEE 802.3 standard defines the standard for the CSMA/CD protocol (the protocol used in Ethernets). The 802.3 CSMA/CD protocol is a random access protocol that has no notion of packet timing constraints or packet priorities. Moreover, because of the probabilistic collision resolution strategy (binary exponential backoff) that it uses, it cannot guarantee a bounded channel access time for any node. Thus the 802.3 standard MAC protocol cannot be used to support the real-time communication services that we mentioned above. The IEEE 802.4 standard represents the standard for the token-passing bus protocol. The 802.4 protocol can guarantee bounded channel access times for packets that belong to the highest priority class. Another attractive feature of the 802.4 protocol is that it provides the ability to handle multiple classes of traffic in an integrated manner using a uniform token-passing paradigm. However, the support provided by the 802.4 standard for real-time communication is limited to these features. Even though the 802.4 standard supports the notion of multiple (limited to a maximum of 4) classes of traffic corresponding to multiple priority levels, there is no support for *global packet level priority resolution*; it is possible with the 802.4 protocol for a lower priority class packet to be transmitted, when packets belonging to higher priority classes are waiting at other nodes (for example, this would happen in a given token cycle, if the token rotation rate in the previous cycle was high enough). The 802.4 protocol also has no notion of individual packet level timing constraints, and hence is blind to the timing requirements of individual packets.

The deficiencies pointed out above make these existing standards for medium access control protocols inadequate for supporting RTCOS and RTCLS. In this chapter, we propose a suite of five window medium access control protocols, viz., PRI, RTDG, RTVC, INTPVC and INTPDG, that is devoid of these deficiencies of the 802.3 and 802.4 standards. The PRI window protocol implements packet-level priority-based arbitration, a capability that the standard MAC protocols lack. The RTDG window protocol takes individual packet-level timing constraints into account in arbitrating access to the channel and has performance characteristics superior to those of the standard protocols. The RTVC window protocol guarantees bounded

channel access times, and has performance characteristics that are comparable to the standard token-passing bus protocol. INTPVC and INTPDG are integrated protocols that support both RTCLS and RTCOS traffic in a unified manner on a common bus, using a uniform window splitting paradigm. We analyze these new protocols and derive various properties exhibited by the protocols. For example, we derive upper bounds on the worst case per packet overhead of the window protocols (Property 6), on the probability that the RTDG window protocol fails to implement the minimum laxity first policy exactly because of new arrivals during a contention resolution interval (Property 10), on the worst case packet service times for the RTVC (Property 11), INTPVC (Property 13) and INTPDG window protocols (Property 14). We also present results of a simulation study conducted in order to evaluate the performance of the protocols. The main properties of the protocols, derived from the analytical and simulation study are summarized below:

1. The per packet contention resolution overhead for the window-based contention resolution procedure used by the window protocols increases logarithmically with the size of the window.
2. The PRI window protocol not only provides a capability, viz., system-wide packet level priority-based arbitration, that the 802.3 and 802.4 standard protocols lack, but also is characterized by a slightly smaller average priority resolution overhead than other non-standard protocols that have been proposed for priority resolution.
3. The RTDG protocol closely approximates the minimum laxity first policy for channel arbitration.
4. The performance of the RTDG window protocol is superior to that of an idealized baseline protocol that provides a bound on the performance achievable by any non-real-time protocol (i.e., a protocol that has no notion of packet timing constraints).

5. The performance of the RTVC window protocol is close to that of an idealized baseline protocol that provides a bound on the performance achievable by any protocol that can guarantee bounded channel access times.
6. The integrated window protocol INTPVC is characterized by a smaller (by a factor of about 2) worst case channel access time for real-time virtual circuit packets, than the standard 802.4 token bus protocol operating in integrated mode (priority mode).
7. The quality of service provided to real-time virtual circuit packets by the integrated window protocol INTPVC is better than that provided by VCTIMER, an idealized baseline integrated protocol.
8. The integrated window protocol INTPDG is characterized by a worst case channel access time for real-time virtual circuit packets, that is about the same as (but larger by about two packet transmission times) than that for VCTIMER.
9. Use of an integrated protocol results in an improvement in the quality of performance. INTPVC provides a better quality of performance for real-time virtual circuit packets than RTVC. INTPDG provides a better quality of performance for real-time datagram packets than RTDG.

In addition to overcoming deficiencies of existing standards, and to the good performance characteristics pointed out above, the protocol suite proposed in this chapter also has the advantage of structural homogeneity, since all the protocols are based on a uniform window splitting paradigm.

The presentation in this chapter is organized as follows. In Section 5.2, we present a homogeneous approach to MAC layer protocols based on a uniform window splitting paradigm. In Section 5.3 through Section 5.6, we employ this approach to develop the window protocol suite for supporting RTCOS and RTCLS. We conclude the chapter in Section 4.7 with a brief summary.

5.2 A Uniform Approach to MAC Protocols

In the preceding sections, we pointed out the limitations of existing standards for MAC protocols and noted that they are not suitable for supporting RTCOS and RTCLS. In this section, we develop a suite of structurally homogeneous MAC protocols that can support both RTCOS and RTCLS, based on a uniform window-splitting medium access control paradigm.

The starting point for this work was provided by the MLF window protocol proposed by Zhao, Stankovic and Ramamritham [108]. This protocol can be used to closely approximate the system-wide minimum laxity first policy for message transmission over a shared bus. Thus, it is well-suited to supporting RTCLS. However this protocol cannot guarantee bounded channel access times for nodes. Therefore this protocol, as it is, is not suitable for implementing RTCOS. An approach that makes use of the window splitting paradigm to implement RTCOS also is attractive for the following reason. In addition to its intuitive appeal, the structural homogeneity that such an approach provides, makes it possible to use the same medium access control logic and LAN controller hardware that is used to support RTCLS, to support RTCOS also. This advantage is especially important when both classes of services are to be supported by a single protocol on a common bus in an integrated manner. This motivated us to explore generalizations and adaptations of the window protocol proposed in [108] that can guarantee bounded channel access times.

We have approached the problem of developing a homogeneous suite of window protocols through a technique known as *parameter-based contention resolution* (PBCR). The term PBCR describes the following channel arbitration problem. Consider a set of N nodes sharing a multiple access channel. Each node is associated with a parameter $p \in \Pi = [0, p_{max}]$ (Π is referred to as the parameter space). The parameter p may vary with time. The problem is to allocate the channel to the node that has the smallest value of p , whenever there is contention for the channel. We reduce the problems of priority-based arbitration, real-time datagram arbitration and real-time virtual circuit arbitration to instances of PBCR and realize PBCR

using a window splitting approach. Each of these reductions gives rise to a window MAC protocol, that implements the corresponding type of arbitration.

In the following sections, we propose a suite of five window MAC protocols, viz., PRI, RTDG, RTVC, INTPVC and INTPDG. The protocol PRI implements priority-based arbitration. The protocol RTDG implements real-time datagram arbitration by implementing a system-wide minimum laxity first policy for transmission. The protocol RTVC is a window protocol characterized by bounded channel access times, and thus may be used to implement real-time virtual circuits. The protocols INTPVC and INTPDG are integrated window protocols that handle both RTCLS and RTCOS packets in a unified manner on a single bus. The protocol INTPVC displays a favorable bias towards servicing real-time virtual circuit packets, while the protocol INTPDG is biased towards servicing real-time datagram packets.

In the next section, we describe a window splitting procedure called window that implements parameter-based contention resolution. In the subsequent sections, we derive the PRI, RTDG, RTVC, INTPDG and INTPVC protocols with this procedure as a basis.

5.2.1 Window Splitting Procedure

Window protocols and other members of the splitting algorithm family like stack and tree protocols (e.g., [12], [16], [50], [58], [99]) have in the past been used as *collision resolution schemes* in multiple access networks; when a collision occurs, these protocols provide a means of ensuring that *all* the packets involved in the collision get transmitted successfully. The window splitting procedure window described below makes a *deliberate* use of collisions and collision detection to identify a *single* node that has the smallest value of a parameter p and assigns transmission rights on the channel to this node. Thus it functions like a *distributed channel scheduler* that chooses the next packet to send.

The procedure window assumes a slotted bus of slot length equal to τ , the maximum round trip propagation delay on the bus. All the nodes are assumed to

be slot-synchronized and are permitted to start transmitting only at the beginning of a slot. The nodes are assumed to have carrier sensing and collision detection capabilities. This CSMA/CD capability enables the nodes to determine whether there was a successful transmission, a collision or silence on the channel during the immediately preceding slot.

The procedure operates as follows. Each node maintains a data structure known as a window that is characterized by the position π of its left edge and its size δ . The window at any moment spans the range of values $[\pi, \pi + \delta)$. This range is referred to as the current window. If the parameter p associated with a node lies in the current window, the node is said to be in the window. The window splitting procedure is invoked at the beginning of each transmission epoch with a current window of $[0, \delta_{max})$. This window is referred to as the initial window. The size of the initial window δ_{max} is chosen to be a power of 2 and it represents the maximum size the window can ever assume. When the window splitting procedure is invoked, if there is only one node in the initial window, then that node continues to transmit its packet to completion; if two or more nodes lie within the window, a collision results, which is sensed by all the nodes at the end of the slot. On detecting a collision, each node splits the window into two halves. The left half window is made the current window and the procedure operates on this window exactly as it did with the initial window, i.e., if there is only one node in the left half window, this node transmits its packet to completion; if there is a collision, the left half is again split into two, and the operation continues recursively until some node transmits successfully. If there are no nodes in the left half window, then there is no transmission on the channel and an idle slot is sensed by all the nodes. In this case, the right half window is made the current window and the procedure operates on this half window in exactly the same manner as it did on the left half.

Figure 5.1 illustrates the operation of the window splitting procedure. In the example shown in this figure, the initial window spans the range $[0, 128)$ and there are three nodes with p values 75, 99 and 120 respectively that are in the window. All of them start transmitting, and consequently there is a collision on the channel. The nodes sense the collision and split the window into two, making the left half of the

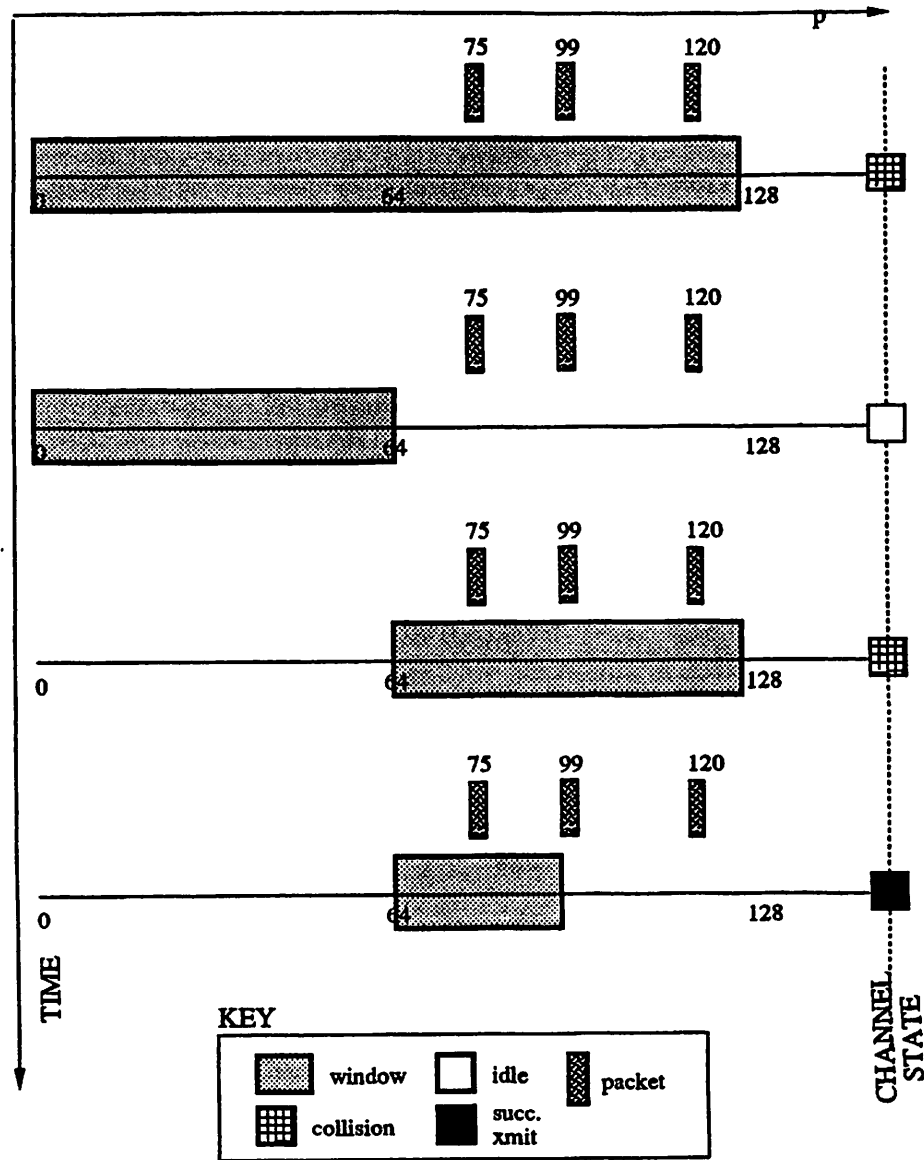


Figure 5.1 An example of the window procedure in operation

```

1. window ( $\Pi, \Delta, R$ )
2.   begin
3.      $\delta \leftarrow \delta_{max} \leftarrow \text{ceil2}(\Delta); \pi \leftarrow 0; \sigma \leftarrow \text{Right};$ 
4.      $p \leftarrow R(\Pi, \Delta);$ 
5.     forever do
6.       if  $p \in [\pi, \pi + \delta)$ 
7.         then start transmitting packet and monitor the channel;
8.         else monitor the channel;
9.       endif
10.      At the beginning of the next slot
11.      event  $\leftarrow$  channel event during the past slot;
12.      switch (event)
13.        case successful transmission:
14.          if I am the transmitting node
15.            then continue to transmit packet to completion;
16.            else wait for transmission to complete;
17.          endif
18.          return;
19.        case collision:
20.           $\sigma \leftarrow \text{Left};$ 
21.           $\delta \leftarrow \frac{\delta}{2};$ 
22.          if  $(\delta < 1)$  then  $\text{window}(\Omega, A, adr);$ 
23.          break;
24.        case idle:
25.          if  $\sigma = \text{Right}$  then return;
26.           $\sigma \leftarrow \text{Right};$ 
27.           $\pi \leftarrow \pi + \delta;$ 
28.          break;
29.      end switch
30.    end forever;
31.  end window.

```

Figure 5.2 The window splitting procedure

initial window the current window. Since none of the nodes lie within this half, the nodes sense the channel to be idle. The right half is then made the current window. All the nodes lie within the current window once again, resulting in a collision. The current window is therefore split into two and the left half of this window considered first. Only one node, the node with the smallest value of p , lies within this half and it transmits its packet to completion. Transmission of the packets with p values 90 and 120 will be scheduled by subsequent invocations of the window procedure.

5.2.2 Pseudocode

Figure 5.2 contains the pseudocode for the window splitting procedure window, executed by each node. The procedure has three arguments Π , Δ and R . The argument Π defines the parameter space under consideration. The argument $\Delta \leq |\Pi|$ is used to compute¹ (line 3) the size δ_{max} of the initial window, so that the size is an exact power of 2. This argument would typically be specified as the size of the parameter space Π . The argument R is a function that defines a rule for the computation (line 4) of the parameter p associated with a node. An example of such a rule (assuming that the parameter space Π is the set of all node addresses) is:

Rule adr: Choose the parameter p associated with the node to be the node's unique address.

The parameter value computed by the rule R is required to be either in the range $[0, \Delta)$ or be undefined (e.g., when the node does not have a packet to transmit). This requirement ensures that the initial window spans all possible values of p .

The defining variables of the window, viz., δ , π and σ are initialized in line 3. The parameter σ indicates which half of the parent window (the window which was split to produce the window under consideration) is being examined and is defined to have the value "Right" for the initial window. The parameter p itself is computed in line 4. Between lines 5 and 30, the procedure essentially implements a recursive binary² partitioning of the range of values of p covered by the initial window $[0, \delta_{max})$. The entire window is examined first (line 6). If there are no nodes in this window (line 25), then none of the nodes are contending for the channel and the procedure therefore returns. If there is exactly one node in the window (line 13), then that node is the one with the lowest p and is given exclusive channel access rights until it completes transmission (lines 14,15,16). If there is more than one

¹The function ceil2 is defined as $\text{ceil2}(x) \triangleq 2^{\lceil \log_2 x \rceil}$, i.e., it is equal to x rounded up to the nearest power of 2.

²A procedure similar to window is specified in the IEEE 802.4 token bus standard for station addition. However this protocol uses recursive quaternary partitioning.

node in the window (which would result in a collision on the channel, line 19), the window is split into two (line 21) and the left half of the window examined first (line 20). The recursive partitioning continues until a successful transmission takes place (line 15) or if the node with the smallest value of p decides not to transmit its packet (e.g., if the packet deadline has expired) (line 25). Note that if two nodes have the same value of p , i.e., if there is a tie, the recursive partitioning would eventually cause the condition tested on line 22 to be true. When this occurs, the procedure window is recursively invoked, this time with the space of addresses of the nodes involved in the tie (Ω), the size (A) of the set of all node addresses and the rule "adr" specified earlier, as its arguments. Since each node is guaranteed to have a unique address, there cannot be a tie in this recursive invocation and the tied (in the original invocation) node with the smallest address successfully transmits its packet.

5.2.3 Properties of window

PROPERTY 5 *The procedure window implements PBCR.*

PROOF: Whenever there is a collision on the channel, the procedure window splits the window into two halves and *always first examines the left half window*, i.e., the half window spanning the smaller values of p . Thus the node ultimately selected for awarding transmission rights has a value of p that is smaller than that of all the other nodes. Therefore the procedure implements PBCR. Q.E.D.

PROPERTY 6 *In the absence of a tie, an upper bound on the worst case per packet contention resolution overhead ξ_{max} (in units of τ) for window is given by:*

$$\xi_{max} \leq 2 \log_2 [\Delta] - 1, \quad (5.1)$$

when $\Delta > 1$. The equality holds good when Δ is an exact power of 2.

PROOF: Initially, let us assume that Δ is a power of 2. Let

$$\Delta = 2^k, \quad k > 0.$$

The result may be proved by induction on k .

Basis Step: When $k = 1$, $\Delta = 2$. Therefore, there can be at most two nodes in the initial window, since $\delta_{max} = \text{ceil}_2(\Delta) = 2$ (line 3 of Figure 5.2) and we have assumed that there are no ties. The worst case overhead occurs when there are two nodes in the initial window. In this case, the node with the smaller value of p will be selected for granting transmission rights immediately after the initial collision, i.e., the worst case overhead is 1 slot, which agrees with Eq. (5.1).

Induction Step: Assume Eq. (5.1) is true for $k = r$, i.e., assume that

$$\xi_{max} = 2r - 1,$$

when $\Delta = 2^r$. When $k = r + 1$, the size of the initial window is doubled and the previous initial window becomes the left half of the new initial window. The worst case contention resolution overhead occurs, when there is no node whose p value falls in the left half of this new initial window, and the worst case contention resolution overhead is incurred in resolving the contention in the right half of this new window. The overhead is given by the sum of the overhead (1 slot) that arises as a result of the collision in the initial window, the overhead (1 slot) that arises as a result of the idle slot involved in examining the left half of the initial window, and the overhead ($2r - 1$ slots, by the inductive hypothesis) in resolving the contention in the right half of the initial window. This sum evaluates to $2r + 1$ slots which is equal to $2k - 1$ slots for $k = r + 1$. Thus Eq. (5.1) is again satisfied.

If Δ is not a power of 2, then $\Delta < \delta_{max}$. The window procedure specifies that the parameter computation rule R has to choose the value of the parameter p associated with a node to be less than Δ . This implies that $p < \delta_{max}$. Therefore only a portion of the right half of the initial window will be examined by the contention resolution procedure. Hence the maximum contention resolution overhead will be smaller than if Δ is a power of 2. In fact, in this case the worst case overhead

incurred in examining the left half of the initial window fully, may be larger than that incurred from examining the 'sparsely filled' right half. However the worst case overhead then is given by the sum of the overhead (1 slot) that arises as a result of the collision in the initial window, and the overhead ($2r - 1$ slots) in resolving the contention in the left half of the window, i.e., $2r = 2k - 2$ slots. This again agrees with inequality (5.1). Q.E.D.

COROLLARY 1 *If a tie is possible, an upper bound on the per packet contention resolution overhead ξ_{max} (in units of τ) for window is given by:*

$$\xi_{max} \leq 2(\lceil \log_2 \Delta \rceil + \lceil \log_2 A \rceil) - 2, \quad (5.2)$$

when $\Delta > 1$. Here A is the size of the space of addresses of all the nodes.

The proof of this property is similar to that of Property 6, except that the worst case contention overhead incurred in order to resolve a tie is included.

PROPERTY 7 *Assuming that the per packet contention resolution overhead is uniformly distributed over its range of values, i.e., that all possible values of the overhead are equally likely, the average per packet contention resolution overhead $\bar{\xi}$ is given by:*

1. $\bar{\xi} \leq \lceil \log_2 \Delta \rceil - 0.5$, if there can be no ties.
2. $\bar{\xi} \leq \lceil \log_2 \Delta \rceil + \lceil \log_2 A \rceil - 1$, if ties are possible.

PROOF: The proof follows from Property 6, Corollary 1, the fact that the minimum possible overhead is 0 slots (when there is no contention), and the assumption that the overhead is uniformly distributed. Q.E.D.

The procedure window may be used to implement a priority resolution protocol in a straightforward manner. We consider this in the next section.

1. forever do
2. window(Φ , K, pri);
3. end forever

Figure 5.3 The PRI window protocol

5.3 The PRI Window Protocol

Figure 5.3 contains the pseudocode for a window protocol PRI that may be used for priority-based arbitration. The priority associated with a node at any instant of time is assumed to belong to the space of priority values $\Phi = [0, K)$. The protocol merely invokes the window contention resolution procedure repeatedly with arguments Φ , K and pri. The parameter computation rule "pri" is defined below:

Rule pri: Choose parameter p to be the priority of the packet with the smallest priority queued at the node. If there are no queued packets p is undefined.

PROPERTY 8 *The PRI window protocol always selects for transmission the packet with the smallest priority value (at the time p is evaluated) in the entire system.*

PROOF: The result follows from Property 5 and the definition of "pri".

It is clear from Section 4.6.1.1 and the discussion in the previous section that the PRI window protocol has a smaller average contention resolution overhead (Property 7) than other priority resolution protocols that have been proposed for bus-based systems.

A real-time datagram arbitration protocol may be implemented using the window splitting procedure in the same manner as PRI is implemented. We consider this protocol, the RTDG window protocol, in the next section.

1. forever do
2. window(Λ, L, mlf);
3. end forever

Figure 5.4 The RTDG window protocol

5.4 The RTDG Window Protocol

5.4.1 Pseudocode

The pseudocode for the RTDG window protocol is shown in Figure 5.4. This protocol repeatedly invokes the window contention resolution procedure with the space of possible laxity values (Λ), the width (L) of the subspace $[0, L) \subset \Lambda$, that determines the initial window size and that is used in computing the parameter p associated with the node, and a rule called "mlf", as its arguments. The parameter computation rule "mlf," is defined below.

Rule mlf: Choose parameter p to be the laxity of the packet with the smallest non-negative laxity at the node. If this laxity is larger than L , or there are no queued packets, then p is undefined (i.e., the node does not contend for the channel). Delete any packet that has a negative laxity from the queue.

The first clause in this rule ensures that the packet with the shortest time to extinction, i.e., the most urgent packet, at each node contends for the channel. The second clause ensures that the initial window in the procedure window spans all possible values of the parameter p (this clause implies that only packets whose laxities are less than L , i.e., packets with a minimum level of urgency can contend for access to the channel; however, since the laxity of a packet continuously decreases with time, the laxity of every packet will eventually become less than L). The third clause stipulates that packets whose deadlines have expired be dropped at the source.

5.4.2 Properties of RTDG

PROPERTY 9 *The RTDG window protocol selects for transmission the packet with the smallest laxity (at the time p is evaluated) in the entire system, if it selects any packet at all.*

PROOF: The result follows from Property 5 and the definition of rule “mlf”.

Property 9 essentially states that the RTDG window protocol approximates a system-wide minimum laxity first policy for packet transmission (whenever it selects a packet for transmission). The protocol approximates, rather than exactly implement the minimum laxity first policy, because the protocol selects for transmission the packet with the smallest laxity *at the time p is evaluated*. It may be seen from Figure 5.2 that there is an interval of time (contention resolution interval) between the time that the parameter p is evaluated (line 4) and the transmission of the packet with the smallest value of p actually starts. If a packet with a smaller laxity arrives during this interval, it is not considered for transmission until one packet transmission time later. However the approximation to the minimum laxity first policy provided by the RTDG window protocol is quite good because the length of the contention resolution interval (Property 7) is small. This may be seen from the following property and example.

PROPERTY 10 *If the length of the contention resolution interval is uniformly distributed, i.e., all values of the contention overhead are equally likely to occur, then the probability that the RTDG protocol fails to exactly implement the minimum laxity first policy (whenever it selects a packet for transmission) is bounded by:*

$$\Pr[RTDG \neq MLF] \leq \frac{\lambda \xi_{max}}{2} + o(\xi_{max}) \approx \lambda([\log_2 L] + [\log_2 A] - 1),$$

under the Poisson arrival assumption. Here ξ denotes the length of the contention resolution interval, ξ_{max} is the maximum length of the contention resolution interval, λ is the mean arrival rate, and A is the size of the space of node addresses.

PROOF: A Poisson process is defined as a counting process in which the probability that 0, 1 or more arrivals occur in an interval of time is given by:

$$\Pr [n \text{ arrivals in an interval } \delta t] = \begin{cases} 1 - \lambda \delta t + o(\delta t) & \text{if } n = 0 \\ \lambda \delta t + o(\delta t) & \text{if } n = 1 \\ o(\delta t) & \text{if } n > 1 \end{cases}$$

Therefore, if the length ξ of the contention resolution interval (CRI) is κ , then the probability that one or more packets arrive during this interval is given by $\lambda \kappa + o(\kappa)$. Let E denote the event, "the laxity of at least one packet that arrives during a CRI (given that packets arrive during the CRI) is smaller than the laxity of the packets currently contending for the channel," and assume that E is independent of the arrival process. By the theorem of total probabilities,

$$\begin{aligned} & \Pr [RTDG \neq MLF] \\ &= \Pr [E] \Pr [\text{packets arrive during a CRI}] \\ &= \Pr [E] \sum_{\kappa=0}^{\xi_{max}} \Pr [\text{packets arrive during a CRI of length } \xi | \xi = \kappa] \Pr [\xi = \kappa] \\ &= \Pr [E] \sum_{\kappa=0}^{\xi_{max}} (\lambda \kappa + o(\kappa)) \Pr [\xi = \kappa], \end{aligned}$$

where ξ denotes the length of the contention resolution interval, ξ_{max} is the maximum length of the contention resolution interval and λ is the mean arrival rate. The result then follows from Property 7 and the uniform distribution assumption, if we make the worst case assumption that that $\Pr [E] = 1$, i.e., at least one of the packets that arrive during a contention resolution interval always has a laxity that is smaller than that of all the packets currently contending for the channel. Q.E.D.

We apply this property below to demonstrate that the probability of deviation from the minimum laxity first policy because of the arrival of new packets is typically small. Consider a system with 32 nodes, an initial window size (L) of 1024, an average load of 0.5, and in which the ratio (α) of the round trip propagation delay to the packet size is 0.01 (a typical value for CSMA protocols). An average load

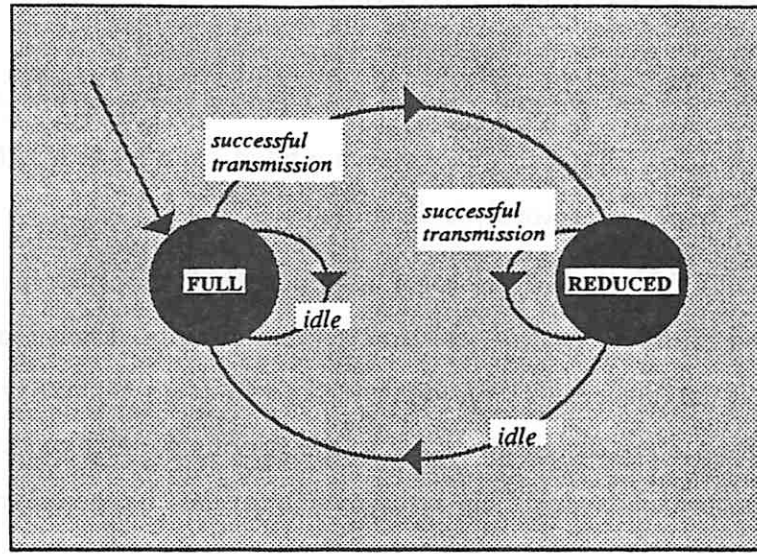


Figure 5.5 RTVC protocol state machine

of 0.5 implies that one packet arrives every $2P$ units of time on the average, where P is the packet length measured in time units and the unit of time is a round trip propagation delay τ . Therefore $\lambda = \frac{1}{2P}$ packets per unit time = 0.005 packets per unit time (here we have made use of the fact that $\alpha = 0.01$).

$$\Pr[RTDG \neq MLF] \leq \frac{\xi_{max}}{2} \leq 0.070.$$

In Chapter 6, we present the results of a performance evaluation via simulation of the RTDG window protocol. These results indicate that the performance of the RTDG window protocol is close to that of an idealized protocol that implements the minimum laxity first policy for channel arbitration with zero overhead, and is superior to the performance of protocols that have no notion of packet timing constraints.

5.5 The RTVC Window Protocol

In this section, we make use of the window splitting procedure window to construct a protocol that can guarantee bounded channel access times. This protocol, the RTVC window protocol, is described below.

The RTVC window protocol assumes that the system consists of a predefined fixed number N_{rtvc} of real-time virtual circuits. Each real-time virtual circuit (RTVC) is associated with an identifier known as its *capability value*. The real-time virtual circuits are distributed among the various nodes by assigning sets of capability values to each node. This assignment may be changed dynamically by network management entities, as the relative requirements at each node change. A node is permitted to make use of a real-time virtual circuit, i.e., it has the capability to make use of it, only if it possesses the capability value of the real-time virtual circuit - hence this terminology. The space of capability values $\Gamma_0 = [0, N_{rtvc})$ is referred to as the full space. In order to guarantee bounded channel access times, this space is shrunk and restored dynamically as part of the protocol operation. This will become clear below. The space of capability values (Γ) under consideration at any instant is denoted Γ and is referred to as the current space.

Figure 5.5 contains a finite state machine description of the RTVC window protocol. The protocol may exist in one of two operational states, FULL and REDUCED. In either state, each node executes the window splitting procedure window. The protocol is initially in state FULL, in which $\Gamma = \Gamma_0$, the full space. When some node completes successful transmission of a packet (by executing window), the protocol enters the state REDUCED, and each node reduces the current space of capability values, by deleting from it the capability value of all the real-time virtual circuits whose capability values are less than that of the real-time virtual circuit whose packet was just transmitted. A real-time virtual circuit whose capability value was deleted is said to have been *disabled*. Disabling a real-time virtual circuit has the effect of temporarily denying the packets belonging to it, the right to contend for access to the channel, and is the key to guaranteeing bounded channel access times. The protocol remains in this state until no node transmits a packet,

```

1.   $C \leftarrow \text{ceil2}(|\Gamma_0|)$ ;  $\Gamma \leftarrow \Gamma_0$ ; state  $\leftarrow$  FULL;
2.  forever do
3.      switch (state)
4.          case FULL:
5.          case REDUCED:
6.              window( $\Gamma$ ,  $C$ , cap);
7.              event  $\leftarrow$  successful transmission ? idle;
8.              switch (event)
9.                  case successful transmission:
10.                      $t \leftarrow$  RTVC whose packet was transmitted;
11.                     foreach  $c \in \Gamma$ 
12.                         if ( $c \leq t$ ) then  $\Gamma \leftarrow \Gamma - \{c\}$ ;
13.                     end foreach;
14.                     state  $\leftarrow$  REDUCED;
15.                     break;
16.                  case idle:
17.                      $\Gamma \leftarrow \Gamma_0$ ;
18.                     state  $\leftarrow$  FULL;
19.                     break;
20.              end switch;
21.          end switch;
22.  end forever

```

Figure 5.6 The RTVC window protocol

at which point it returns to state FULL. This can happen if either (i) no real-time virtual circuit has a packet waiting to be transmitted or (ii) all the real-time virtual circuits are disabled. The pseudocode presented next provides the details.

5.5.1 Pseudocode

Figure 5.6 contains the pseudocode for the RTVC window protocol. The protocol is first initialized (line 1). The protocol then consists of repeatedly (lines 2,22) executing procedure window (line 6) and the appropriate state transitions (lines 14,18). The procedure window is called with the current space (Γ), the size (C) of the full space and the rule “cap” specified below, as its arguments.

Rule cap: Choose parameter p to be the capability value of the enabled real-time virtual circuit, that has the smallest capability value among all the enabled real-time virtual circuits at the node, that have a packet to transmit.

Note that if there are multiple real-time virtual circuits assigned to a node, only one of them (the one that has the lowest capability value among the currently enabled real-time virtual circuits assigned to the node) is permitted to contend for channel access. Depending on what event (line 7) the last execution of window resulted in, the protocol switches (line 8) to the appropriate next state. If there was a successful transmission by some node, then all the nodes switch to the REDUCED state (line 14). In addition, all the real-time virtual circuits whose capability values are less than or equal to the capability value of the real-time virtual circuit, whose packet was just transmitted are disabled (lines 11-13). This is done by deleting their capability values from Γ (line 12), so that they will not be considered in the evaluation of p in subsequent invocations of the window procedure (see Rule cap). If the channel was idle instead (line 16), then Γ is restored to the full space Γ_0 (line 17) and the protocol switches to the FULL state (line 18), effectively reinitializing itself.

5.5.2 Properties of RTVC

PROPERTY 11 *An upper bound on the service time S of a packet (defined as the sum of the channel access time and the packet transmission time) is given by*

$$S \leq N_{rtvc}(P + \xi_{max}),$$

where P is the packet length measured in units of τ and ξ_{max} is the upper bound on the contention resolution overhead for the window procedure window provided by Property 6.

PROOF: Consider a real-time virtual circuit with a capability value $k \in [0, N_{rtvc})$.

If it is *enabled*, then in the worst case it will get a chance to transmit its packet within k packet transmission times. This is true because, a real-time virtual circuit is disabled immediately after it transmits a packet. Thus even if all the real-time virtual circuits with capability values less than k have several packets to transmit, the protocol will permit at most one packet to be transmitted from each of these real-time virtual circuits. Therefore the worst case packet service time is given by

$$S \leq k(P + \xi_{max}) \leq N_{rtvc}(P + \xi_{max}).$$

The term $(P + \xi_{max})$ in the inequalities above represents an upper bound on the packet transmission time, given by the sum of the packet length and the upper bound on the worst case contention resolution overhead provided by Property 6.

On the other hand, suppose it is *disabled*. It can be in the disabled state for at most $N_{rtvc} - k$ packet transmission times. This is true because, once an RTVC is disabled only RTVCs with larger capability values are permitted to contend for the channel (lines 11-13 of Figure 5.6 and Rule cap). In the worst case, all of these $N_{rtvc} - k - 1$ RTVCs may have packets to transmit. However, since they are permitted to transmit at most one packet each and disabled thereafter, the system will consist wholly of disabled RTVCs after $N_{rtvc} - k - 1$ packet transmission times. At this point, since all the nodes sense the channel to be idle, the nodes revert to the FULL state and all the RTVCs are re-enabled. By the argument given in the previous paragraph, the RTVC with capability value k will have to wait for at most k packet transmission times more to transmit its packet. Thus the worst case packet service time is given by

$$S \leq N_{rtvc} (P + \xi_{max}).$$

Q.E.D.

The existence of an upper bound on the packet service time guaranteed by Property 11, makes the RTVC window protocol appropriate for implementing real-time virtual circuits. However, how does the worst case packet service time for this protocol compare with other protocols? Property 12 derived below is intended to provide a comparison of the relative values of the worst case packet service time for the RTVC window protocol and for an arbitrary protocol PROT for which the worst case service time is given by $N_{rtvc} (P + \beta)$ (See Section 4.6.1.2). The worst case packet service time for most protocols that have a bounded channel access time is given by this last expression. For example, for a token-passing protocol, β corresponds to the token passing overhead; for an idealized hybrid random access protocol that incurs no overhead, and TDMA, $\beta = 0$.

PROPERTY 12 *The fractional increase (f) in the upper bound on the packet service time because of employing the RTVC window protocol instead of a protocol PROT which is characterized by an upper bound on the channel access time of $N_{rtvc}(P + \beta)$, $\beta \geq 0$, is given by*

$$f = \frac{\alpha(\xi_{max} - \beta)}{\tau + \alpha\beta}.$$

This property trivially follows if we note that

$$f = \frac{N_{rtvc}(P + \xi_{max}) - N_{rtvc}(P + \beta)}{N_{rtvc}(P + \beta)},$$

and that $P = \frac{\tau}{\alpha}$.

As an example, consider a system with 32 real-time virtual circuits and $\alpha = 0.01$. Compared to a protocol for which $\beta = 0$, the fractional increase in the upper bound on the packet service time is less than 0.09, by the above property. Here we have made use of Property 6, and the fact that a tie cannot develop in the window procedure, because real-time virtual circuit capability values are unique. If we consider a system of 1024 real-time virtual circuits instead, this fractional increase is less than 0.19.

We present the results of a performance evaluation by simulation of the RTVC window protocol in Section 6. These results indicate that, in spite of the small fractional increase in the worst case channel access time that we noted above, the performance of the RTVC window protocol (measured in terms of the performance measure of interest, viz., the *guarantee ratio* or the fraction of packets that arrive that actually are accepted) is close to that of an idealized protocol with $\beta = 0$. This is to be expected since the guarantee ratio is mainly dependent on the worst case channel access time, and is not significantly affected unless the worst case channel access time increases by an amount that is of a magnitude similar to that of packet laxities. But packet laxities have to be greater than the worst case channel

access time, in order for packets to be accepted for transmission. Therefore, fractional increases in the worst case channel access time do not significantly affect the performance.

The RTVC window protocol may be used to support RTCOS, while the RTDG window protocol may be used to support RTCLS. In the next section, we consider the integrated protocols that may be used to support both RTCOS and RTCLS in a unified manner on a single bus.

5.6 Integrated MAC Protocols

In the last two sections, we examined MAC protocols that may be used to support RTCOS and RTCLS. These protocols are suitable for supporting either RTCOS or RTCLS, but not both. In this section, we examine integrated protocols that may be used to support both classes of services in a unified manner.

Integrated protocols have been considered in the context of multimedia communication in which diverse classes of traffic such as voice, video, facsimile and standard data are handled in a unified manner [50]. Token-passing protocols have been extended using timers to support multiple classes of traffic. The IEEE 802.4 token bus protocol operating in priority mode [33] and the FDDI protocol ([79], [80]) are examples of standard integrated protocols. Both these protocols are based on a token-passing paradigm. The 802.4 token bus protocol is a standard for *bus-based* systems, while the FDDI protocol has been proposed as a standard for high speed *token rings*. These protocols are timed token rotation protocols that permit lower priority messages (known as *asynchronous* messages in FDDI terminology) to be transmitted, only if recent movement of the token among nodes has been sufficiently fast relative to a target token rotation time (TTRT). It has been shown that this protocol can guarantee an upper bound on the channel access time for high priority messages (*synchronous* messages in FDDI terminology) of $2 * TTRT$ [83].

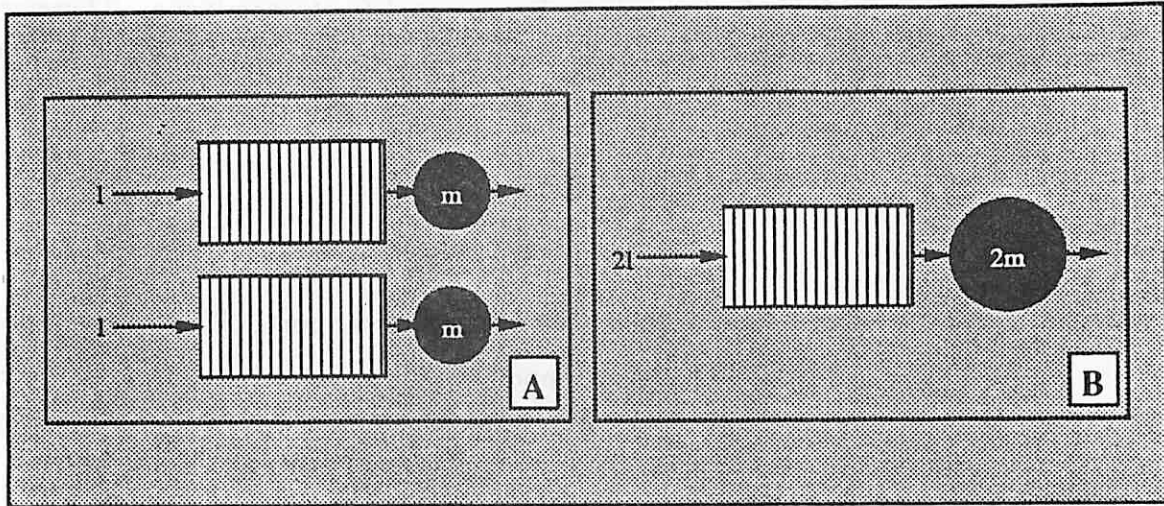


Figure 5.7 Two queuing scenarios

If each node is permitted to transmit a maximum of one synchronous packet per token rotation then $TTRT = N * P$, where N is the number of nodes and P is the packet transmission time. Thus the worst case bound on the channel access time for high priority or synchronous messages in this case is $2 * N * P$.

Integrated handling of multiple classes of traffic has a number of advantages, related to both *physical implementation* and *performance* [76]. In terms of physical implementation, an integrated approach permits the use of a common interface and common cabling for all the classes of traffic. In addition, such an approach also results in conservation of chassis space. In order to provide fault-tolerance, a distributed system architecture incorporates redundant components ([89], [59]). If there are k classes of traffic, each associated with r slots (corresponding to r redundant channels) in the chassis housing a node, then the number of slots required can be reduced from kr to r through the use of an integrated protocol. The *performance advantage* offered by integration is illustrated by the following example that employs simple queueing theoretic arguments.

Consider a system with two classes of traffic. The packets in each class of traffic arrive according to a Poisson process with an average arrival rate of l . Further each packet requires a service time of s . Figure 5.7 illustrates two ways of handling the

two classes of traffic. In scenario A, each stream of traffic has a server of capacity $m = \frac{1}{2}$ dedicated to servicing its packets. In scenario B, the two streams are merged into a single stream, and the two servers are merged into a single server of capacity $2m$. Assuming that the two streams of traffic are independent, the stream produced by merging them also is a Poisson stream, but with an average arrival rate $2l$. Note that the average load (ρ), defined as the ratio of the average arrival rate to the average service rate is identical and equal to $\frac{l}{m}$ in both scenarios.

In an M/G/1 queue, the average delay (W) suffered by a customer is given by the Pollaczek-Khinchtin equation,

$$W = E[X] + \frac{\lambda E[X^2]}{2(1-\rho)}, \quad (5.3)$$

where X is the service time for a customer, λ is the average arrival rate and ρ is the average system load. For an M/D/1 system with service time S , since the variance of the service time is zero, this reduces to

$$W = S \left(1 + \frac{\rho}{2(1-\rho)} \right) = K_\rho S, \quad (5.4)$$

where K_ρ is a constant independent of S . The queues involved in our example may be modeled as M/D/1 queues with service times s (Scenario A) and $\frac{1}{2}s$ (Scenario B). The average waiting times W_A and W_B for scenarios A and B are given by

$$W_A = K_\rho s \quad \text{and} \quad W_B = \frac{1}{2} K_\rho s.$$

Thus the average delay experienced by packets in the non-integrated scenario is twice that for the integrated scenario. This drastic difference in performance is to be expected, since Scenario A permits a channel to be idle even when there are packets queued on the other channel. In Scenario B, packets do not have to wait if there are no packets already in the queue, since there is only one queue.

The M/D/1/FCFS model of the above example is not an appropriate model of real-time virtual circuit operation or real-time datagram arbitration. Also, the

performance measure of average delay is only a rough indicator of the quality of performance of real-time virtual circuit operation or real-time datagram operation. However, it is reasonable to expect that the underlying reason for the improvement in performance (namely the efficient utilization of the channel bandwidth by making available the bandwidth unused by one class of traffic to the other class) will produce performance improvements, for systems with other queueing disciplines and other performance measures (that are affected by the average delay). The potential advantages that integration has to offer motivate the following goal, namely to develop new integrated medium access control layer protocols that can be used to support the two classes of traffic, viz., RTCOS and RTCLS traffic, that arise in a distributed real-time system, in a unified manner on a common bus. We have developed two integrated protocols, INTPVC and INTPDG in Section 5.6.1 and Section 5.6.2 respectively. INTPVC displays a bias towards real-time virtual circuit packets, while INTPDG displays a bias towards real-time datagram packets. Unlike the FDDI protocol and the token bus protocol, which are blind to the individual packet timing constraints of asynchronous messages, these protocols explicitly consider the timing constraints of best effort messages. Further the worst case channel access time associated with the INTPVC protocol is about half that for the token bus and FDDI protocols (Section 5.6.1); the worst case channel access time associated with the INTPDG protocol is about the same as that associated with the token bus and FDDI protocols (Section 5.6.2).

5.6.1 INTPVC

The INTPVC window protocol extends the RTVC window protocol to utilize idle channel time (when there are no real-time virtual circuit packets to transmit) for the transmission of real-time datagrams. Figure 5.8 contains a finite state machine description of the protocol. Note that the state machine for the INTPVC protocol is identical to that of the RTVC window protocol except for the introduction of a new state labeled "RTDG" and transitions into and out of this state. Whenever there is no real-time virtual circuit packet to transmit (all the real-time virtual

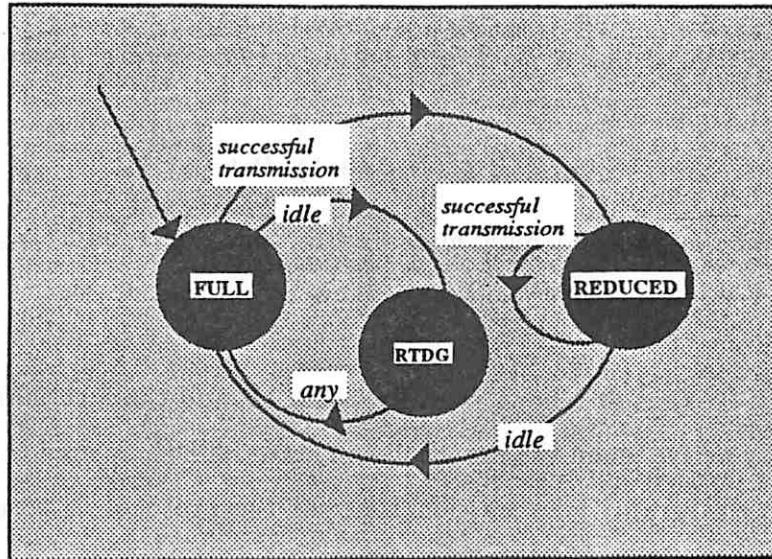


Figure 5.8 INTPVC protocol state machine

circuits are enabled, but there is no transmission, i.e., the protocol state is FULL and the channel event is "idle") the protocol shifts to the state "RTDG". In this state, the node executes the procedure window for real-time datagram arbitration and then returns to the state FULL. Note that real-time datagrams are considered for transmission, *only when there are no real-time virtual circuit packets pending for transmission*. Thus this protocol is biased towards the servicing of real-time virtual circuit packets.

5.6.1.1 Pseudocode

Figure 5.9 contains the pseudocode for this protocol. The code essentially describes the operations of the state machine depicted in Figure 5.8 and is self-explanatory. Note that the window procedure is called with the arguments relevant to real-time virtual circuit arbitration in lines 5 and 20 (states FULL and REDUCED), while it is called with the arguments relevant to real-time datagram arbitration in line 35 (state RTDG).

5.6.1.2 Properties of INTPVC

PROPERTY 13 *An upper bound on the service time S of a real-time virtual circuit packet is given by*

$$S \leq (N_{rtvc} + 1)(P + \xi_{max}),$$

where P is the packet length measured in time units and ξ_{max} is the worst case contention resolution overhead for window.

PROOF: This result follows from Property 11, and the fact that a real-time virtual circuit may arrive when the protocol is in state RTDG. It may take up to one packet (real-time datagram) transmission time for the protocol to return to state FULL. Q.E.D.

Thus, in spite of integration with real-time datagram arbitration, the channel access time for real-time virtual circuits is still bounded, although it has increased slightly.

5.6.2 INTPDG

The INTPDG window protocol, like the INTPVC protocol, extends the RTVC window protocol to utilize idle channel time (when there are no real-time virtual circuit packets to transmit) for the transmission of real-time datagrams. In addition, it also partitions the available bandwidth equally between real-time datagrams and real-time virtual circuit packets, without dedicating any portion of the bandwidth to either. This is illustrated in Figure 5.10, which contains the state machine diagram for the protocol. Compared to Figure 5.8, this figure contains a new state "RTDG-R" and new transitions into and out of this state. Also, the state RTDG has been relabeled RTDG-F. The window procedure is invoked to perform real-time virtual circuit arbitration, whenever the protocol is in states FULL or REDUCED; it is used to perform real-time datagram arbitration, whenever the protocol is in states RTDG-F or RTDG-R. Note that the protocol enters the state RTDG-R after every successful

real-time virtual circuit packet transmission. In this state, up to one real-time datagram may be transmitted, if there are any in the system. Since every real-time virtual circuit packet transmission may be followed by the transmission of a real-time datagram, real-time datagrams may use up to half the available channel bandwidth. In addition, any unused real-time virtual circuit bandwidth (state RTDG-F) is also made available to real-time datagrams. Thus this protocol is biased towards real-time datagrams.

5.6.2.1 Pseudocode

Figure 5.11 contains the pseudocode for the INTPDG protocol. The code essentially describes the operations of the state machine depicted in Figure 5.10 and is self-explanatory.

5.6.2.2 Properties of INTPDG

PROPERTY 14 *An upper bound on the service time S of a real-time virtual circuit packet is given by*

$$S \leq (2N_{rtvc} + 1)(P + \xi_{max}),$$

where P is the packet length measured in time units and ξ_{max} is the worst case contention resolution overhead for window.

PROOF: Note that, if the REDUCED and RTDG-R states are coalesced into a single state, and the packet transmission time in the resultant state is doubled (to account for the fact that the transmission of each real-time virtual circuit packet may be followed by the transmission of a real-time datagram), then the resultant state machine becomes equivalent to the state machine in Figure 5.8. The result then follows from Property 13. Q.E.D.

Thus, in spite of integration with real-time datagram arbitration, the channel access time for real-time virtual circuits is still bounded, although it has approximately doubled.

5.7 Discussion

This section contains a discussion of various aspects related to the window protocols presented in this chapter.

5.7.1 Bandwidth Exchange

The RTVC window protocol and the integrated protocols that incorporate it, partition the total channel bandwidth into a number of logical channels (RTVCs) that have bounded packet delivery times. Each of these channels may be treated as an independent resource that can be scheduled independently of others and that can be exchanged between nodes. This ability to segment the available channel bandwidth into independent channels is useful in the end-to-end scheduling of tasks. The ability to exchange RTVCs between nodes is also useful. For example, if one node has a dearth of RTVCs but a sustained higher level of traffic, it can possibly improve its guarantee ratio by obtaining RTVCs from another node that has a surplus of RTVCs. This ability to exchange RTVCs may be referred to as *bandwidth exchange*, since the channel bandwidth previously held by one node is now made available to another node. This transfer of RTVCs between nodes is achieved by transferring the corresponding capability values. A node transfers an RTVC to another node by sending the capability value associated with the RTVC to the transferee and deleting the capability value from its own records (in effect, forgetting it). One can think of different schemes to manage the exchange of RTVCs between nodes. One scheme would be to use a centralized RTVC server that manages all the free RTVCs in the system. A node that needs a RTVC can borrow one from the RTVC server and return any surplus RTVCs in its possession to the server. Another scheme would be to use a bidding algorithm to locate a node that has sufficient surplus of RTVCs and send a request for RTVCs to it.

5.7.2 Ring Topology

The window protocols that we have developed belong to the class of CSMA/CD protocols and hence assume a bus topology. Distributed real-time systems based on a bus topology are popular in real-time applications. For example, the Distributed Systems Data Bus (DSDB) and the Real-Time Communications Network developed by IBM FSD in Manassas, Virginia are based on a bus topology. The IEEE 802.4 standard for the token-passing bus has been adopted for process control applications. One of the advantages of the bus topology is that there are no active components on the transmission path. A system based on a bus topology is therefore more robust. The ring topology, on the other hand, requires active repeaters at each node. The failure of a repeater can disrupt the entire system. However the ring-based approach has the advantages of being better suited to fiber optic topology (since no taps are required) and a wider geographical coverage. Ring-based networks have recently been studied and proposed as an alternative for real-time networks [94].

Protocols analogous to PRI, RTDG and RTVC can be easily identified for the ring topology. The priority resolution scheme proposed in the IEEE 802.5 standard for token rings, that we described in Chapter 4, provides the same functionality as PRI. Realizing the functionality of the RTDG window protocol requires that the priority and reservation fields in the above scheme be expanded to accommodate laxity values. Yao and Zhao have studied the effect on the performance of mapping laxity values into a set of priority levels for the IEEE 802.5 priority resolution scheme. The IEEE 802.5 token ring protocol operating in nonpriority mode (i.e., as a simple token passing protocol) guarantees bounded channel access times, and hence may be used to implement real-time virtual circuits. Realizing the analogs of INTPVC and INTPDG however is more involved.

It may be seen from Figure 5.8 that INTPVC performs real-time datagram arbitration (i.e., shifts to state RTDG) only when there are no real-time virtual circuit packets to transmit in any node. A token-passing protocol that realizes the functionality of INTPVC is sketched below. The protocol assumes the existence of a distinguished node in the system (say node 1) designated as the RTVC monitor.

Further the token is assumed to include two additional bits called the RTVC bit and the mode bit. The functions of the RTVC monitor, the RTVC bit and the mode bit will become clear shortly. The protocol operates in two modes, viz., the RTVC mode and the RTDG mode. In the RTVC mode, the mode bit is cleared, and access to the ring is arbitrated according to the standard token passing protocol operating in nonpriority mode, i.e., a node that has a packet to transmit waits for the free token to arrive, marks the token busy, transmits its packet, waits for the token and packet to circulate around the ring and return, removes the packet from the ring, marks the token free, and releases it into the ring again. In addition, if some node on the ring has a RTVC packet to transmit, this node sets the RTVC bit in the circulating busy token; also, the node modifies the reservation field of the token appropriately, if the laxity of its least laxity packet is smaller than the value in the reservation field. The reservation and RTVC bits are copied onto the free token by the node which originally marked the token busy when it releases the free token. If the RTVC monitor node receives a free token with the RTVC bit cleared, then none of the nodes had a RTVC packet to transmit during the preceding trip of the token around the ring. In this case, the RTVC monitor sets the mode bit to indicate that the protocol has switched to the RTDG mode. In the RTDG mode, access to the channel is arbitrated according to the standard token passing protocol operating in priority mode. The RTVC monitor copies the reservation field onto the priority field of the token, and the first node that has a packet with a laxity that is smaller than the priority field gets to transmit on the ring. Other nodes update the RTVC bit and reservation fields to reflect the current state of their queues. When this node releases the free token into the ring, it clears the mode bit so that the protocol returns to the RTVC mode. The channel access time for a real-time virtual circuit packet with this protocol is clearly bounded by $(N_{rtvc} + 1)(P + \tau)$, where τ is the token passing overhead, since one real-time datagram packet may be transmitted in addition to N_{rtvc} real-time virtual circuit packets in a token cycle.

A token ring analog of the INTPDG protocol may similarly be realized. It may be seen from Figure 5.10 that the INTPDG protocol considers a real-time datagram for transmission, whenever the system is idle (state RTDG-F) and immediately after

the transmission of each real-time virtual circuit packet (state RTDG-R). The token ring analog of INTPVC described in the previous paragraph can be modified slightly to realize the analog of INTPDG. The mode bit is set by any node that transmits a real-time virtual circuit packet, rather than just by the RTVC monitor. This causes the protocol to shift to the RTDG mode. When the token returns to the node that set the mode bit, the mode bit is cleared and the free token is released again. An additional bit known as the quota bit will be necessary to ensure that utmost one real-time datagram is transmitted before the token returns to the node that set the mode bit. The channel access time for a real-time virtual circuit with this protocol is the same as that for the INTPDG protocol, except that the contention resolution overhead is replaced by the token-passing overhead.

5.8 Summary

We considered a distributed real-time system based on a local area network with a bus topology, and addressed the problem of medium access control layer support for providing real-time connection-oriented service (RTCOS) and real-time connectionless service (RTCLS). We identified MAC layer support for priority resolution, bounded packet service times and explicit consideration of timing constraints as requirements for the implementation of RTCOS and RTCLS. We identified the deficiencies of the existing medium access control layer standards for bus-based systems, viz., the IEEE 802.3 standard and the IEEE 802.4 standard, in their support for RTCOS and RTCLS. We proposed a new suite of protocols that is devoid of these deficiencies.

The proposed suite consists of five protocols, viz., PRI, RTDG, RTVC, INTPVC and INTPDG, all based on a uniform window splitting paradigm. The protocol PRI implements priority-based arbitration, and has an average priority resolution overhead that is smaller than other protocols that have been proposed for priority arbitration in bus-based systems. The protocol RTDG implements real-time datagram arbitration based on the minimum laxity first policy. The protocol RTVC

implements real-time virtual circuit arbitration. The protocols INTPVC and INTPDG are integrated protocols that may be used to support both real-time datagram and real-time virtual circuit arbitration in an integrated manner on a common bus. INTPVC is characterized by a smaller (by a factor of about 2) worst case channel access time for real-time virtual circuit packets, than the standard 802.4 token bus protocol operating in integrated mode. We analyzed the window protocols and derived various properties exhibited by the protocols. We derived upper bounds on the worst case per packet contention resolution overhead (Property 6), the probability that the RTDG window protocol fails to implement the minimum laxity first policy exactly (Property 10), the worst case packet service times for the RTVC window protocol (Property 11), INTPVC window protocol (Property 13) and the INTPDG window protocol (Property 14).

In addition to overcoming the deficiencies of existing standards, and to the good performance characteristics that will become evident in the next chapter, the protocol suite proposed in this paper also has the advantage of structural homogeneity, since all the protocols are based on a uniform window splitting paradigm.

We conclude our description of the window protocols with this summary. In the next chapter, we consider the evaluation of the performance of the window protocols.

```

1.   $\Gamma \leftarrow \Gamma_0$ ; state  $\leftarrow$  FULL;
2.  forever do
3.      switch (state)
4.          case FULL:
5.              window( $\Gamma$ ,  $C$ , cap);
6.              event  $\leftarrow$  successful transmission ? idle;
7.              switch (event)
8.                  case successful transmission:
9.                       $t \leftarrow$  RTVC whose packet was transmitted;
10.                     foreach  $c \in \Gamma$ 
11.                         if ( $c \leq t$ ) then  $\Gamma \leftarrow \Gamma - \{c\}$ ;
12.                     end foreach;
13.                     state  $\leftarrow$  REDUCED;
14.                     break;
15.                  case idle:
16.                      state  $\leftarrow$  RTDG;
17.                      break;
18.              end switch;
19.          case REDUCED:
20.              window( $\Gamma$ ,  $C$ , cap);
21.              event  $\leftarrow$  successful transmission ? idle;
22.              switch (event)
23.                  case successful transmission:
24.                       $t \leftarrow$  RTVC whose packet was transmitted;
25.                      foreach  $c \in \Gamma$ 
26.                          if ( $c \leq t$ ) then  $\Gamma \leftarrow \Gamma - \{c\}$ ;
27.                      end foreach;
28.                      state  $\leftarrow$  REDUCED;
29.                      break;
30.                  case idle:
31.                       $\Gamma \leftarrow \Gamma_0$ ; state  $\leftarrow$  FULL;
32.                      break;
33.              end switch;
34.          case RTDG:
35.              window( $\Lambda$ ,  $L$ , mlf);
36.              state  $\leftarrow$  FULL;
37.              break;
48.      end switch;
39.  end forever

```

Figure 5.9 The INTPVC window protocol

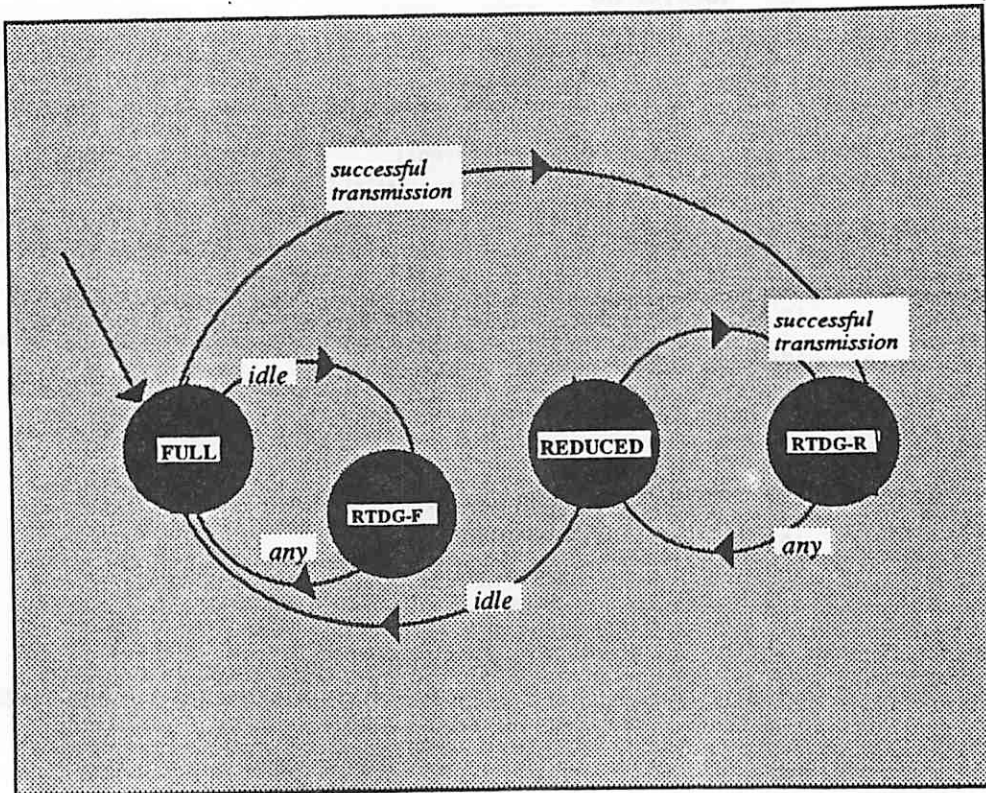


Figure 5.10 INTPDG protocol state machine

```

1.   $\Gamma \leftarrow \Gamma_0$ ; state  $\leftarrow$  FULL;
2.  forever do
3.      switch (state)
4.          case FULL:
5.              window( $\Gamma$ , C, cap);
6.              event  $\leftarrow$  successful transmission ? idle;
7.              switch (event)
8.                  case successful transmission:
9.                       $t \leftarrow$  RTVC whose packet was transmitted;
10.                     foreach  $c \in \Gamma$ 
11.                         if ( $c \leq t$ ) then  $\Gamma \leftarrow \Gamma - \{c\}$ ;
12.                     end foreach;
13.                     state  $\leftarrow$  RTDG-R;
14.                     break;
15.                  case idle:
16.                      state  $\leftarrow$  RTDG-F;
17.                      break;
18.                  end switch;
19.          case REDUCED:
20.              window( $\Gamma$ , C, cap);
21.              event  $\leftarrow$  successful transmission ? idle;
22.              switch (event)
23.                  case successful transmission:
24.                       $t \leftarrow$  RTVC whose packet was transmitted;
25.                      foreach  $c \in \Gamma$ 
26.                          if ( $c \leq t$ ) then  $\Gamma \leftarrow \Gamma - \{c\}$ ;
27.                      end foreach;
28.                      state  $\leftarrow$  RTDG-R;
29.                      break;
30.                  case idle:
31.                       $\Gamma \leftarrow \Gamma_0$ ; state  $\leftarrow$  FULL;
32.                      break;
33.                  end switch;
34.          case RTDG-F:
35.              window( $\Lambda$ , L, mlf);
36.              state  $\leftarrow$  FULL;
37.              break;
38.          case RTDG-R:
39.              window( $\Lambda$ , L, mlf);
40.              state  $\leftarrow$  REDUCED;
41.              break;
42.      end switch;
43.  end forever

```

Figure 5.11 The INTPDG window protocol

CHAPTER 6

PERFORMANCE EVALUATION

In this chapter, we describe a simulation study conducted in order to evaluate the performance of the window protocols developed in Chapter 5. In Section 6.1, we define the performance metrics that are appropriate for evaluating the performance of the MAC protocols considered in this paper. In Section 6.2, we present a set of idealized MAC protocols that serve as baselines for comparison. In Section 6.3, we describe the model of the system that the simulation programs assume. In Section 6.4, we present the results of the simulation study.

6.1 Performance Metrics

A common performance measure used in the evaluation of network protocols is the *average delay* of packets. In the context of real-time communication, the average delay can provide only a rough measure of the quality of the performance of protocols. For example, one can expect that more datagram packets will be delivered within their deadlines, if the average delay is smaller. Similarly, by Little's law [42], one can expect the average lengths of real-time virtual circuit queues to be smaller, when the average delays are smaller. Hence more of the arriving packets are likely to be accepted. However, more direct performance measures are required for real-time communication.

A metric that has been used for measuring the performance of real-time datagram arbitration protocols is the *loss fraction* ([50],[107],[108]). The loss fraction (LF) is defined as:

$$LF = \frac{PL}{PL+PT},$$

where PL is the number of packets that are lost (i.e., that do not meet their deadlines), and PT is the number of packets that are transmitted successfully. Thus the loss fraction measures the fraction of the packets that do not meet their deadlines and may be viewed as a rough measure of the probability that a real-time datagram does not meet its deadline.

A useful metric for measuring the performance of protocols used to support real-time virtual circuits is the *guarantee ratio*. This measure was originally proposed in the context of task scheduling in real-time systems [18] for the measurement of the quality of performance of dynamic task scheduling algorithms. The guarantee ratio is defined as

$$GR = \frac{PG}{PG + PR},$$

where PG denotes the number of packets guaranteed and PR denotes the number of packets rejected. Thus the guarantee ratio measures the fraction of packets that arrive that are accepted, and may be viewed as a measure of the probability that a real-time virtual circuit packet gets guaranteed (and thereby successfully transmitted).

6.2 Baseline Protocols

In this section, we present various baseline protocols that we used in order to perform a comparative evaluation of the protocols presented in this paper. All the baseline protocols that we consider except the TDMA protocol are idealized protocols that cannot be realized in practice. They are idealized in the sense that they do not incur any of the overheads that practical protocols do. The idealized baseline protocols thus provide an upper bound on the performance achievable by any real protocol. The use of an idealized baseline protocol provides the following advantages. First, development of the simulation programs becomes straightforward, since an idealized baseline protocol is simpler to simulate. Second, an idealized protocol provides a simple way of comparing a newly proposed protocol, with a whole class of similar protocols that have already been proposed or that may be proposed

in future, in a single experiment. For example, the 802.3D protocol, the token bus protocol and the virtual token passing protocols that we described in Section 4.6.1.2, all belong to the same class of protocols as does the RTVC window protocol. Rather than compare the RTVC window protocol with each of these protocols, it is sufficient to compare it with a single idealized baseline protocol that provides an upper bound on the performance achievable by any protocol that belongs to this class. If the comparison shows a satisfactory degree of closeness between the performance of the RTVC window protocol and the idealized baseline protocol, then one can conclude that, if at all any performance improvement is achievable by using another protocol instead of the RTVC window protocol, this improvement is not significant. On the other hand, if significant deviations from the idealized baseline protocol are observed, then it would be necessary to compare the RTVC window protocol with the real protocols that belong to the class.

We have made use of four baseline protocols, viz., CML, INRT, IRTVC and VCTIMER in our comparative study. We describe these baselines next.

6.2.1 CML

The CML (centralized minimum laxity first) protocol is a hypothetical protocol in which an omniscient central arbiter, with automatic global knowledge of the laxities of real-time datagram packets in all the nodes, determines which packet will be transmitted next, on the basis of the minimum laxity first policy. The performance of the CML protocol provides a bound on the performance that is achievable by any protocol that attempts to implement the minimum laxity first policy (e.g., RTDG window protocol), since it implements the policy without incurring any form of overhead.

6.2.2 INRT

The INRT protocol is an *idealized non-real-time* medium access protocol. It is a non-real-time protocol in the sense that the protocol does not have any notion of packet timing constraints. In this respect, it is similar to existing standard

medium access control protocols such as the standard Ethernet protocol (IEEE 802.3 CSMA/CD), the standard token-passing bus protocol (IEEE 802.4) and the standard token ring protocol (IEEE 802.5) that are blind to the timing constraints of packets when arbitrating access to the shared channel. The INRT protocol is idealized in the sense that it does not incur any contention overheads that are incurred by other protocols. In essence, it may be described as an idealized token passing protocol that operates without any token-passing delay. Thus at low loads (when very few nodes have packets to transmit and consequently random access protocols incur minimum overhead) it behaves like a random access protocol, while at high loads (when almost every node has a packet to transmit and TDMA incurs minimum overhead) it behaves like a TDMA protocol. Since the INRT protocol incurs no contention overhead whatever the load conditions are, its quality of performance represents an upper bound on the performance of any non-real-time protocol. In other words, the loss fraction corresponding to the INRT protocol is the lowest that can be achieved by any *non-real-time* protocol.

6.2.3 IRTVC

The IRTVC protocol is an *idealized* real-time virtual circuit protocol. It is a real-time virtual circuit protocol in the sense that it can guarantee bounded channel access times. It is idealized in the sense that the per packet contention resolution overhead for IRTVC is zero, and a real-time virtual circuit packet experiences no unnecessary delay when none of the other real-time virtual circuits in the system have a packet to transmit. Operationally, the IRTVC protocol is identical to the INRT protocol that we described above, i.e., it may be described as an idealized token passing protocol that operates without any token-passing delay. We use the IRTVC protocol as a reference for comparing the RTVC window protocol.

It should be noted that even though the TDMA protocol described in Section 4.6.1.2 has no per packet contention resolution overhead, it is still not an ideal protocol at low loads. This is because TDMA is not a random access protocol. At low loads, a node will have to wait for its turn in the round-robin order in order

to transmit its packet, even if none of the other nodes have a packet to transmit. Thus packets are unnecessarily delayed and may cause a low laxity packet (packets with laxities close to (slightly larger than) the worst case channel access time) that arrives at a node to be unnecessarily rejected. It should be pointed out that the RTVC window protocol is a random access protocol.

6.2.4 VCTIMER

The VCTIMER protocol is an idealized integrated protocol that integrates the handling of real-time virtual circuit packets and real-time datagram packets, using a (idealized) token-passing scheme that incurs no token-passing overhead and a timer that keeps track of the rotation time of the token. We use this protocol as a baseline for comparing the performance of the integrated window protocols INTPVC and INTPDG.

The VCTIMER protocol operates in *cycles*. At the beginning of each cycle, a timer called the *token rotation timer* (TRT) is initialized to a value equal to the *target token rotation time* (TTRT). The TTRT for the VCTIMER protocol is equal to the product of the number of real-time virtual circuits (N_{rtvc}) and the packet transmission time (P),

$$TTRT = N_{rtvc}P.$$

A token then circulates among the real-time virtual circuits starting from the first RTVC. When a real-time virtual circuit receives the token it is permitted to transmit up to one packet after which it passes the token onto the next real-time virtual circuit. As mentioned before, the token passing time for the protocol is zero. Each time a packet is transmitted, the TRT is decremented by an amount equal to the transmission time for one packet. If, after all the real-time virtual circuits have had a chance to transmit, the TRT still has not timed out, then the token continues to be passed around to each node as before. But now a node that has the token is permitted to transmit only real-time datagram packets (up to one real-time datagram packet). As before, the TRT is decremented by an amount equal to the transmission time for one packet whenever a real-time datagram packet is transmitted. This

continues until the TRT times out or there are no real-time datagram packets to be transmitted, at which point a new cycle is started.

Note that the VCTIMER protocol is similar to the IEEE 802.4 token bus protocol (in priority mode) and the FDDI protocol. Both the protocols use a token-passing scheme and a token rotation timer to divide the available bandwidth between the class of messages that require an upper bound on the channel access delay (synchronous packets/real-time virtual circuit packets) and the class of messages that do not (asynchronous packets/real-time datagram packets). Like the token bus and FDDI protocols, the VCTIMER protocol is blind to the timing constraints of real-time datagram packets and does not explicitly consider individual packet timing requirements in arbitrating access to the channel. However the VCTIMER protocol, unlike the FDDI protocol, is an *idealized* protocol that makes use of a *centralized* common timer (rather than individual timers at each node) and that incurs no token-passing overheads.

In the next section, we describe the model of the system assumed in the simulation study.

6.3 System Model

For the purposes of simulation, we modeled the system as follows.

1. Time is measured in units of τ , the round trip propagation delay on the channel. The value of τ is 5 μ seconds for a typical local area network with a medium length of 500m (assuming a speed of propagation of electromagnetic radiation in the medium of 2×10^8 m/sec).
2. The system consists of N nodes and N_{rtvc} real-time virtual circuits.
3. The packet length is P bits. The ratio of the round trip propagation delay to the packet transmission time ($\frac{P}{B}$, where B is the transmission bit rate), is denoted as α .

4. Real-time datagram packets arrive at each node, at an average rate of λ_{dg} , according to a Poisson process. The real-time datagram load ρ_{dg} is defined as

$$\rho_{dg} = \frac{N\lambda_{dg}}{\mu}. \quad (6.1)$$

where $\mu = \frac{B}{P}$ is the service rate of the channel and B is the transmission bit rate.

5. Real-time virtual circuit packets arrive at each node, at an average rate of λ_{vc} , according to a Poisson process. The real-time virtual circuit load ρ_{vc} is defined as

$$\rho_{vc} = \frac{N_{rtvc}\lambda_{vc}}{\mu}. \quad (6.2)$$

For the purposes of simulation, packets belonging to all the virtual circuits are assumed to contend for the channel. This is a worst case assumption, since a node may actually possess access capability to several real-time virtual circuits, but at any node packets belonging to only the enabled virtual circuit with the smallest capability value will contend for channel access.

6. The laxities of real-time datagram packets are uniformly distributed in the range $[0, 2\bar{L}_d]$, where \bar{L}_d is the average laxity of the real-time datagram packets. The laxities of the real-time virtual circuit packets are assumed to be uniformly distributed in the range $[0, 2\bar{L}_v]$, where \bar{L}_v is the average laxity of the real-time virtual circuit packets.

The simulation study of the protocols was conducted using discrete-event simulators that incorporate the above model. The programs were written in C and run on a Sequent Balance 3100.

6.4 Simulation Results

The simulation experiments conducted were aimed at examining the performance aspects of the window protocols. In this section, we have summarized the

results of the experiments and compared the performance of the window protocols with the various baseline protocols discussed in Section 6.2. The presentation also includes a study of the performance improvements that may be obtained by using an integrated window protocol instead of using the RTDG and RTVC protocols separately. 95% confidence intervals were computed for the points on the plots; the actual points lie within an interval of width less than 10% of the point estimate of the ordinate. We assumed that $\alpha = 0.01$ (a typical value for CSMA protocols) in all our experiments, corresponding to a packet transmission time of 100 units. Like any other CSMA protocol, we don't expect our protocols to be applicable when at higher values of alpha ($\alpha \geq 0.1$).

In order to keep this presentation short, we have provided only representative plots in this section. These plots summarize the main results of this study. A more comprehensive presentation of the results is provided in Section 6.4.4.

6.4.1 RTDG Window Protocol

The performance of the RTDG window protocol was observed to be insensitive to the number of nodes for a given real-time datagram load (ρ_{dg}). The performance of the protocol was observed to decrease with the window size initially and then reach a minimum after which it remained constant. We used a window size corresponding to the minimum. Figure 6.1 contains a comparison of the performance of the RTDG window protocol with the baseline CML and INRT protocols, measured in terms of the loss fraction. The RTDG window protocol is clearly superior to the INRT protocol whose loss fraction represents a lower bound on the loss fraction achievable by any non-real-time protocol including the existing standards such as the 802.4 token bus protocol and 802.3 CSMA/CD protocol. The performance of the RTDG window protocol may also be observed to be very close to that of the CML protocol. Thus the RTDG window protocol closely implements the global minimum laxity first policy for channel arbitration.

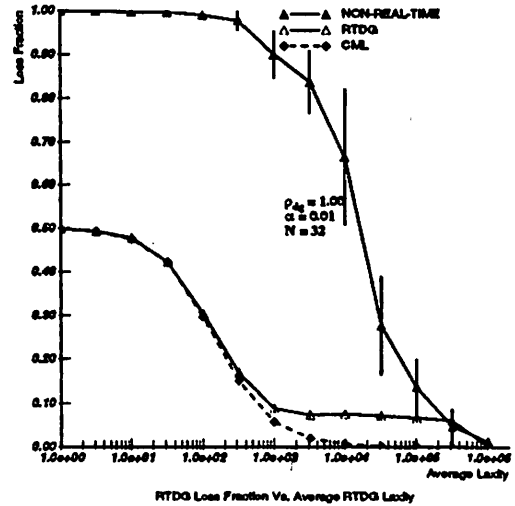
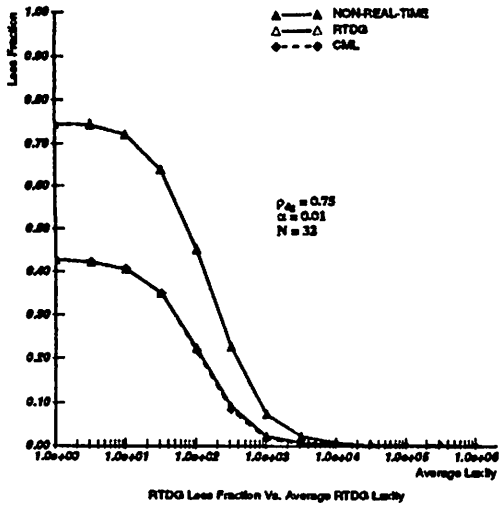
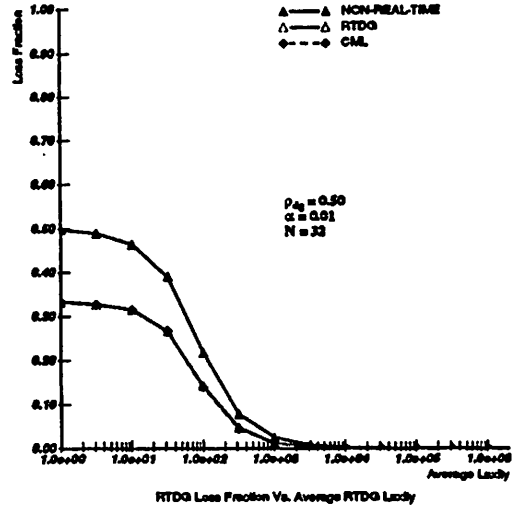
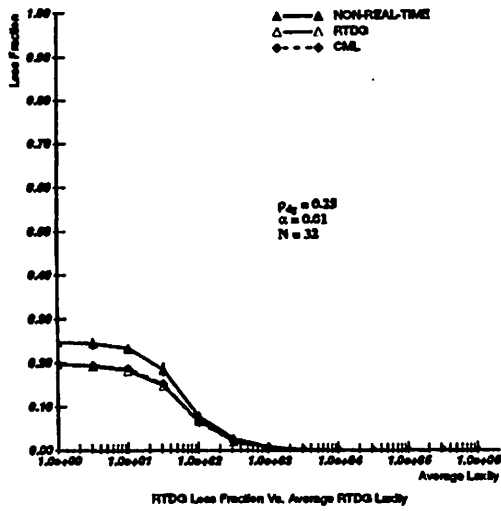


Figure 6.1 Simulation study results: RTDG protocol

6.4.2 RTVC Window Protocol

Figure 6.2 contains a comparison of the performance of the RTVC window protocol with the baseline IRTVC protocol and the TDMA protocol, measured in terms of the guarantee ratio. It may be observed that the performance of the RTVC window protocol is close to that of the idealized baseline protocol (except at very heavy loads). Thus, if some other protocol can provide a performance that is better than that provided by the RTVC window protocol, then the improvement in performance achievable by using the other protocol is not significant. Note that under certain conditions, the performance of TDMA is actually poorer than that of the RTVC window protocol. This happens in Figure 6.2, for example, when $\rho_{vc} = 1$ and the average laxity is less than approximately 30000 time units, and when $\rho_{vc} = 1.5$ and the average laxity is less than 10000 time units. The fact that the performance of TDMA is actually poorer at high load conditions may appear to contradict intuitive expectations. This apparent contradiction may be explained by observing that the effective load that the channel handles is a function of both the actual load and the average laxity of packets. If the average laxity is small, then the number of packets that actually get accepted and queued is small. Thus even if the actual load is high, the effective load in this case will be small.

6.4.3 Integrated Window Protocols

Figure 6.3 contains a comparison of the performance of the integrated window protocols INTPVC and INTPDG with the timer-based baseline protocol VCTIMER. It may be observed that INTPVC, which is biased towards real-time virtual circuit packets, performs better than the VCTIMER protocol, where the performance is measured in terms of the guarantee ratio. Similarly INTPDG, which is biased towards real-time datagram packets, performs better than the VCTIMER protocol, where the performance is measured in terms of the loss fraction. However the performance of the integrated window protocols is poorer than that of the VCTIMER protocol for the class of traffic that it is biased against. For example, INTPVC has

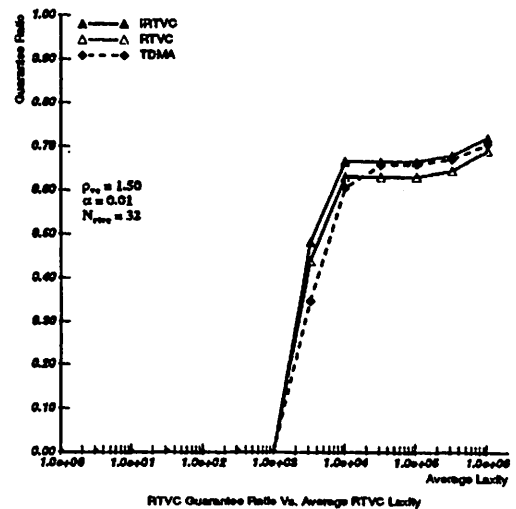
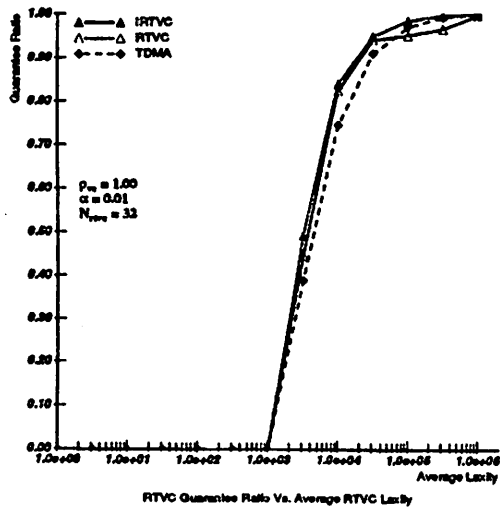
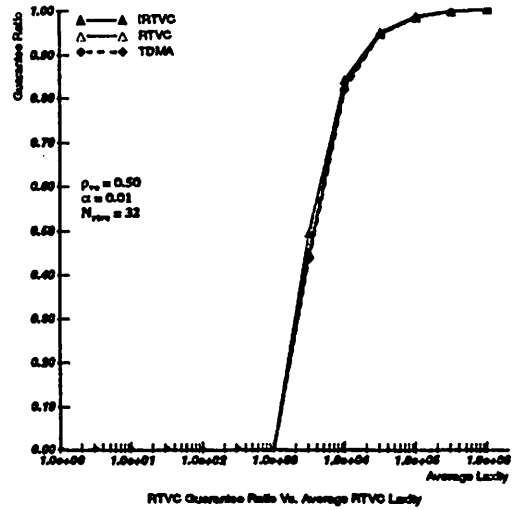
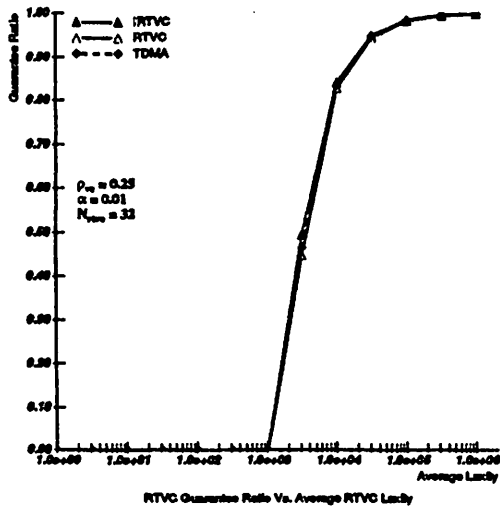


Figure 6.2 Simulation study results: RTVC protocol

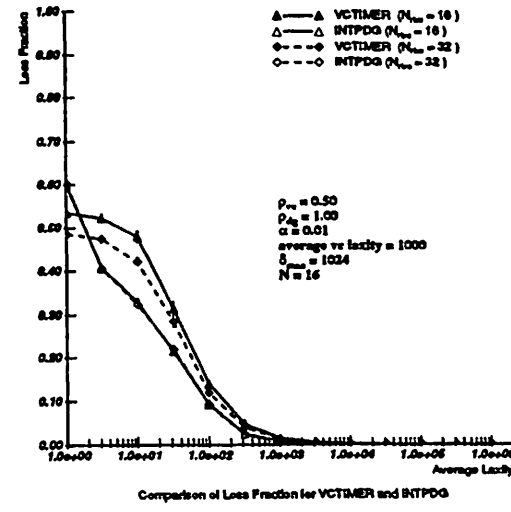
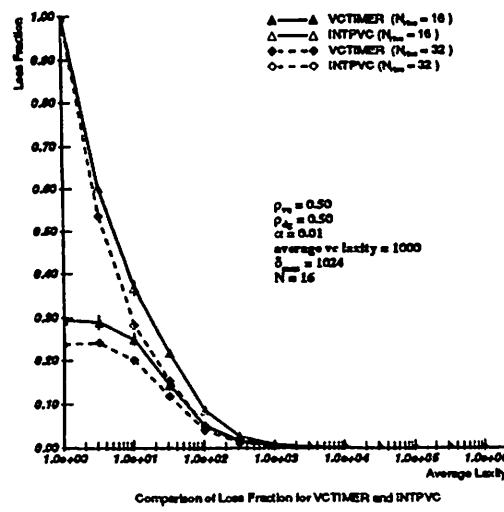
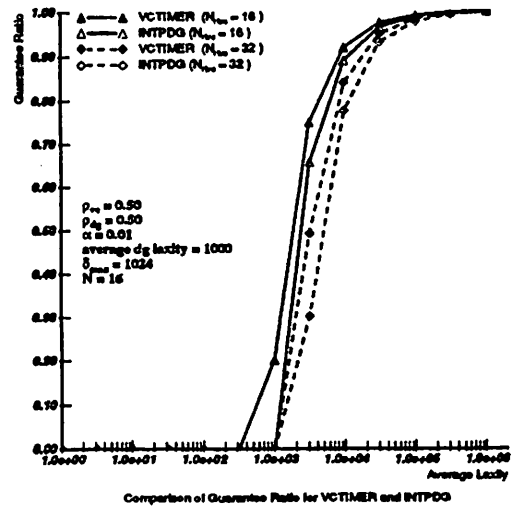
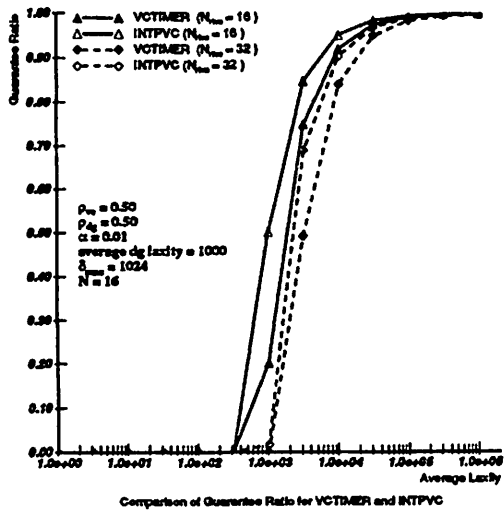


Figure 6.3 Simulation study results: INTPVC and INTPDG

a higher loss fraction than VCTIMER and INTPDG has a higher guarantee ratio than VCTIMER.

Figure 6.4 depicts the effects of integration. The plots contain a comparison of the performance of the integrated protocols and their non-integrated versions, e.g., INTPVC vs RTVC and INTPDG vs RTDG. The use of integrated protocol INTPVC results in improved quality of both real-time datagram and real-time virtual circuit services up to a certain load (which depends on the real-time datagram load). Beyond this value, the quality improvement in real-time virtual circuit services is maintained, but at the expense of the real-time datagram services. Thus INTPVC would be a good choice, if improvement in the quality of service for real-time virtual circuits is desired, while willing to accept a poorer quality of real-time datagram services at higher real-time virtual circuit loads. The use of integrated protocol INTPDG results in improved quality of service for real-time datagram packets (provided the laxity is not too small), but causes some deterioration in the quality of service for real-time virtual circuit packets, especially at low average real-time virtual circuit packet laxities. Thus INTPDG would be a good choice, if improvement in the quality of service for real-time datagram packets is desired while willing to accept some deterioration in the quality of service for real-time virtual circuit packets.

6.4.4 Simulation Results: A Comprehensive Presentation

In this section, we provide a comprehensive presentation of the simulation study results summarized in the previous section. The summary in the previous section was illustrative and considered only representative values of the system parameters. The presentation in this section is more detailed and covers a wide range of parameter values.

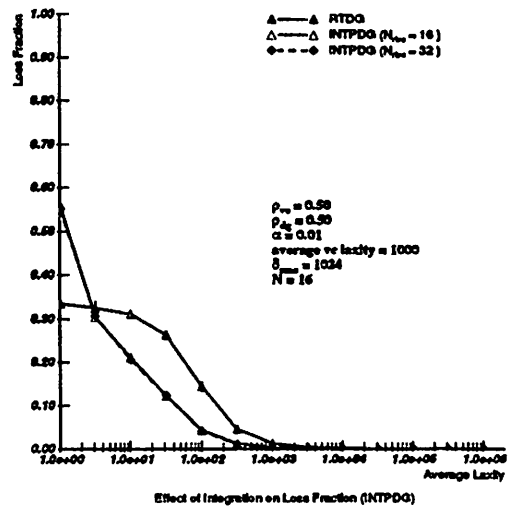
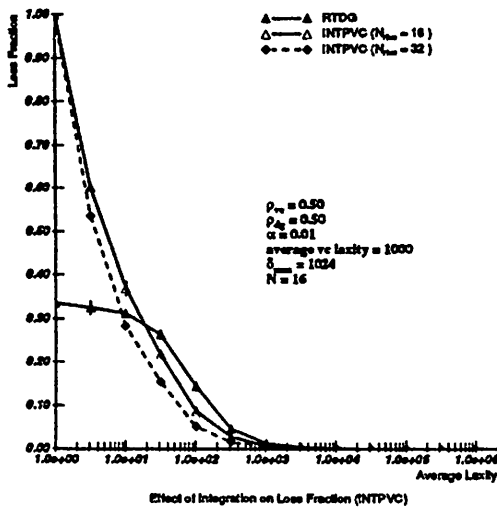
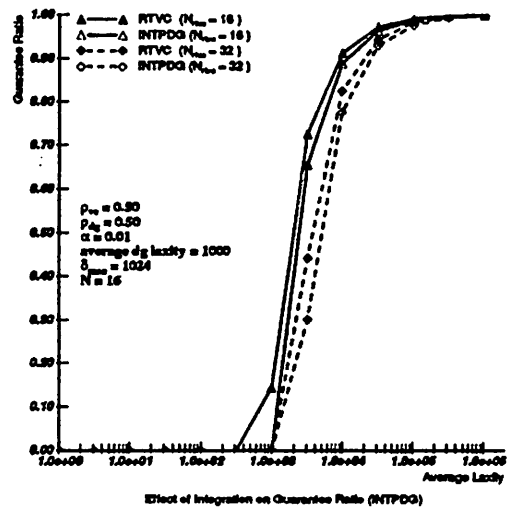
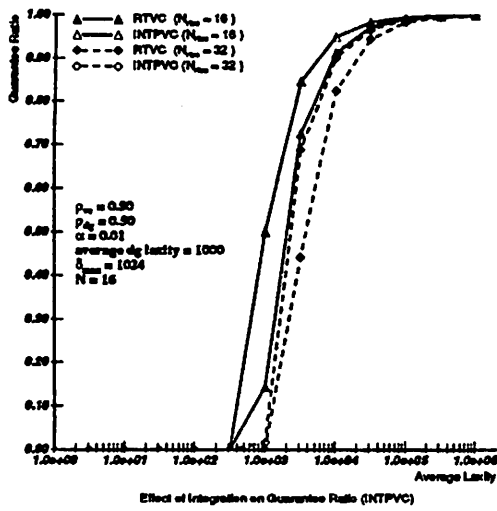


Figure 6.4 Simulation study results: effects of integration

6.4.4.1 RTDG Window Protocol

Dependence of Loss Fraction on Number of Nodes

Figures 6.5, 6.6 and 6.7 depict the dependence of the loss fraction for the RTDG window protocol on the number of nodes (N) in the system, for three different representative values (128, 1024 and 16384) of the maximum window size δ_{max} (measured in time units). It may be seen that, the loss fraction is independent of the number of nodes in the system. This is to be expected, since, for a given load (ρ_{dg}), increasing the number of nodes decreases the arrival rate at each node, i.e., the load is merely redistributed across a larger number of nodes.

Dependence of Loss Fraction on Window Size

Figures 6.8, 6.9 and 6.10 depict the dependence of the loss fraction for the RTDG window protocol on the maximum window size (δ_{max}) used in the protocol, for three different values (16, 32 and 64) of the number of nodes (N). It may be seen that, in general, the loss fraction decreases with increasing window size and approaches a constant value (that depends on the average laxity of packets) beyond a certain window size. This may be explained as follows. Increasing the window size beyond the maximum possible laxity of packets will not result in any additional packets being considered for arbitration and hence will not affect the loss fraction.

Dependence of Waste Fraction on Window Size

The window protocol is a collision-based protocol in which a portion of the channel bandwidth is wasted in contention resolution. We measure this wastage in terms of a secondary performance measure known as the *waste fraction*. The waste fraction is defined as the fraction of the utilized (non-idle) channel time spent on contention resolution. Figures 6.11, 6.12 and 6.13 depict the dependence of the waste fraction for the RTDG window protocol on the maximum window size (δ_{max}) used in the protocol, for three different values (16, 32 and 64) of the number of

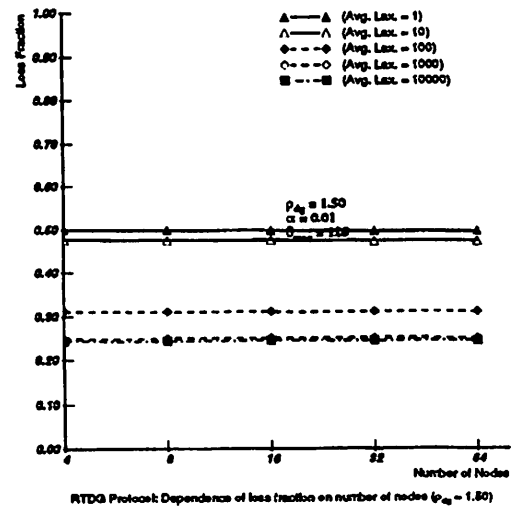
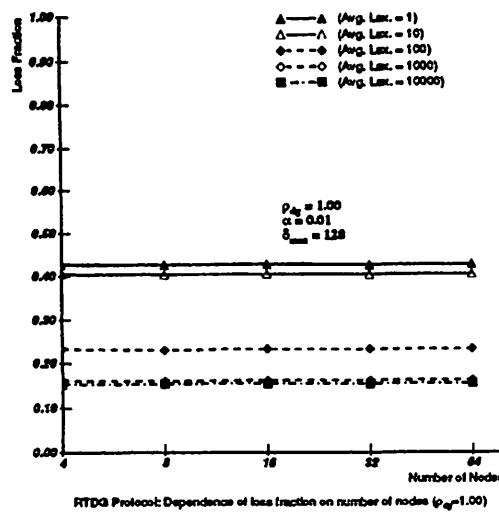
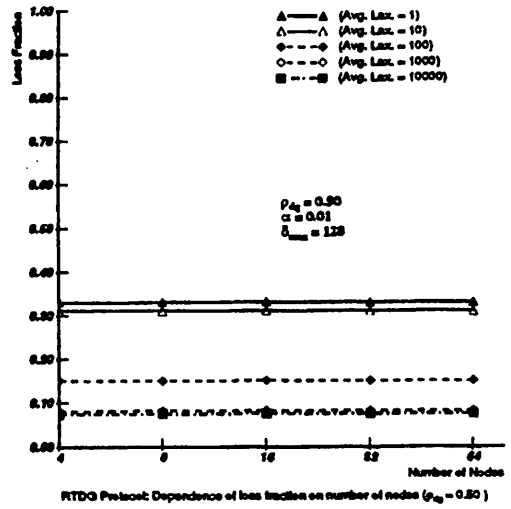
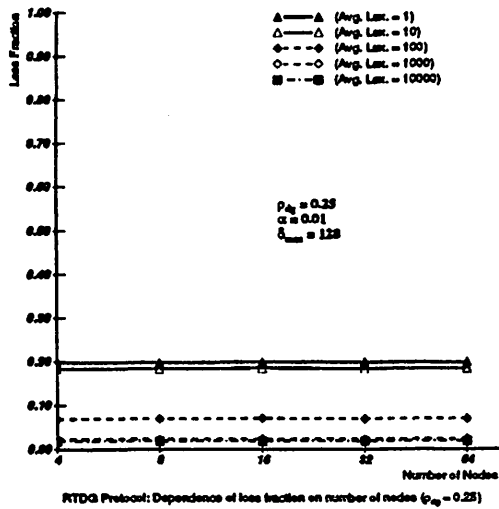


Figure 6.5 Dependence of loss fraction on number of nodes ($\delta_{max} = 128$)

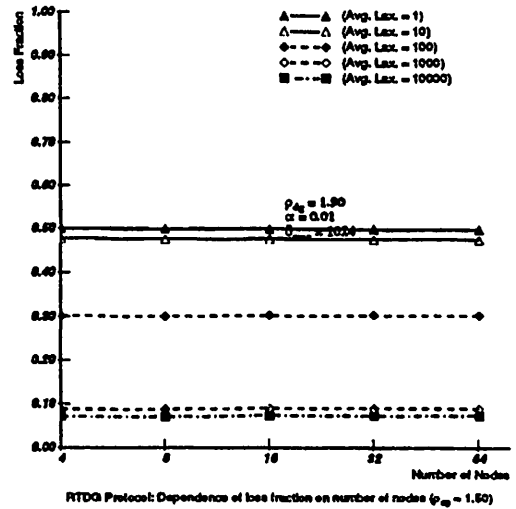
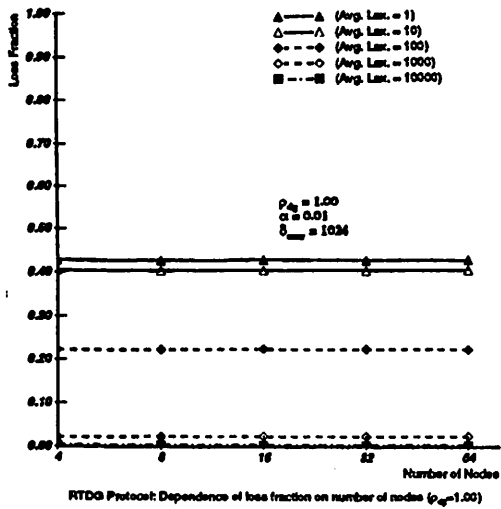
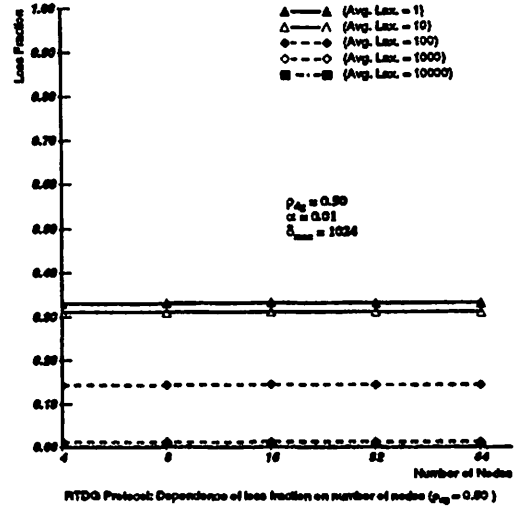
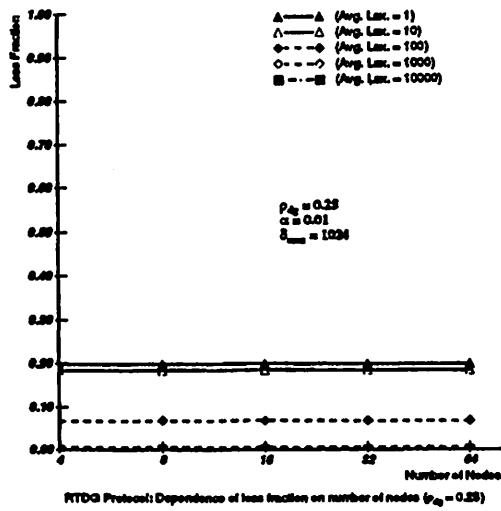


Figure 6.6 Dependence of loss fraction on number of nodes ($\delta_{max} = 1024$)

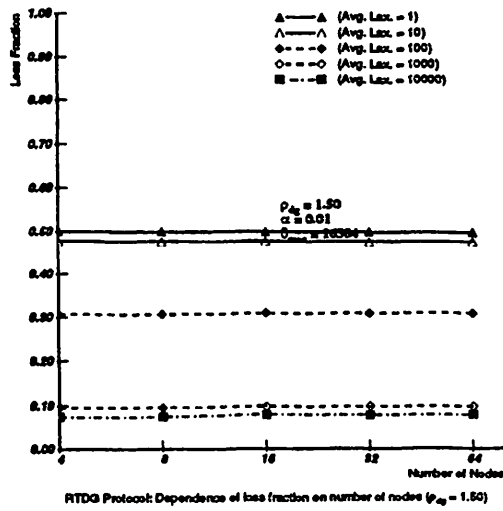
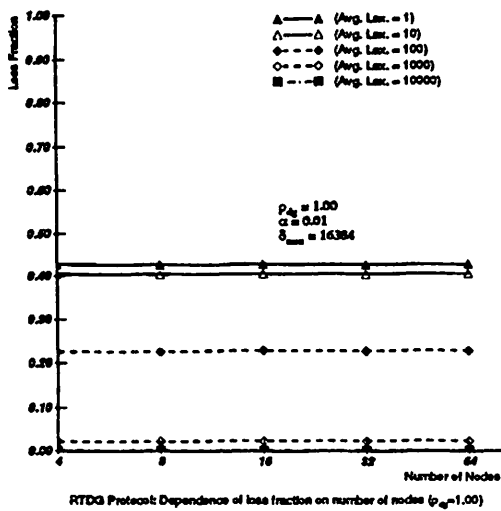
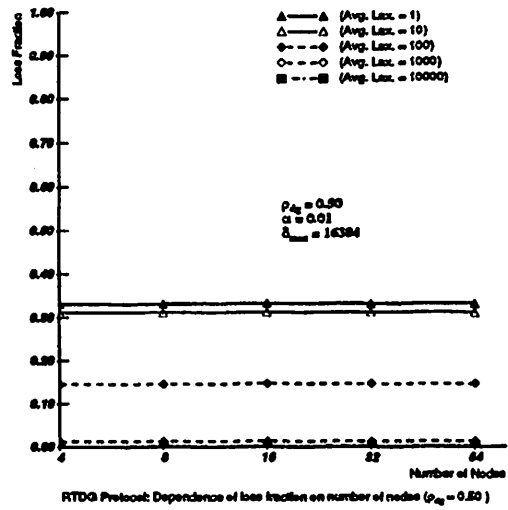
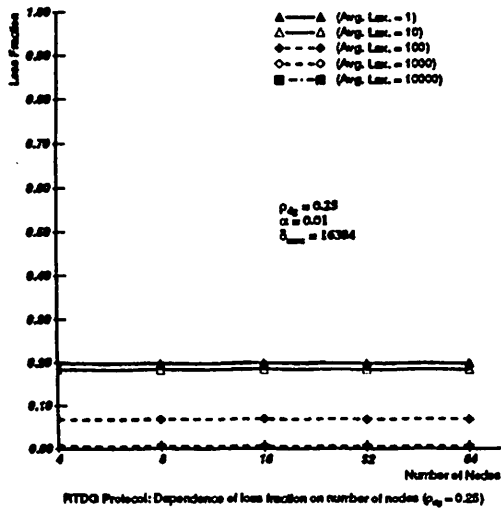


Figure 6.7 Dependence of loss fraction on number of nodes ($\delta_{max} = 16384$)

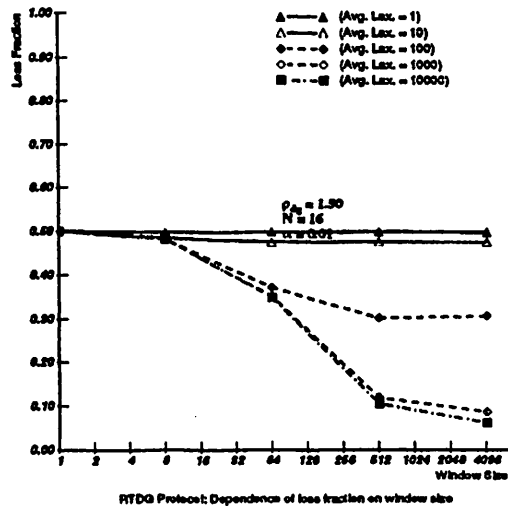
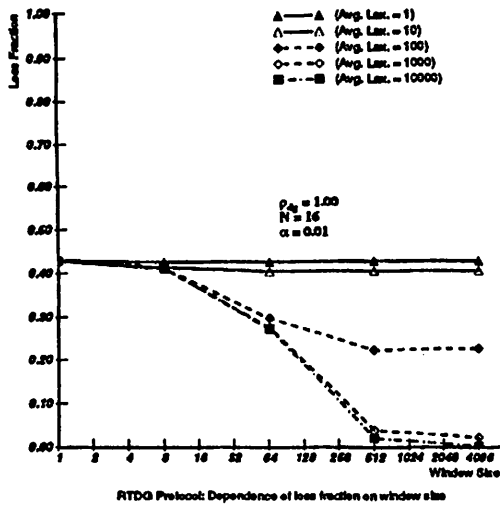
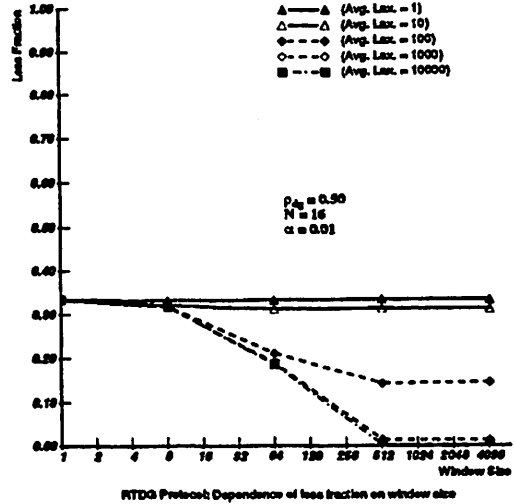
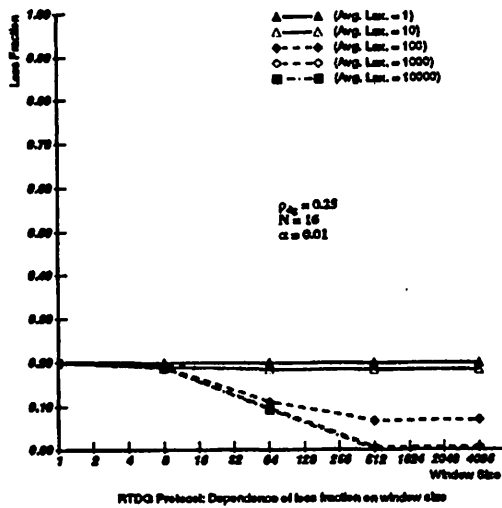


Figure 6.8 Dependence of loss fraction on window size ($N = 16$)

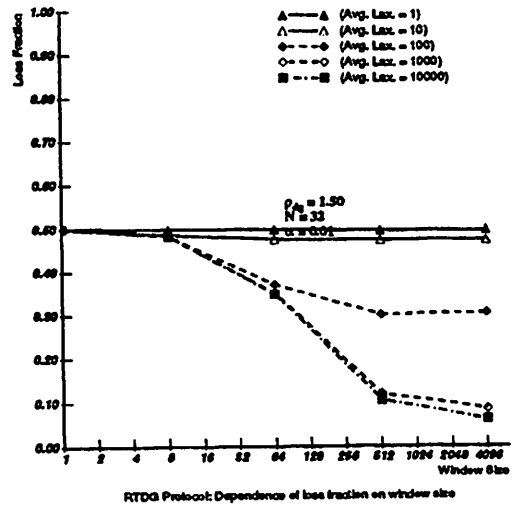
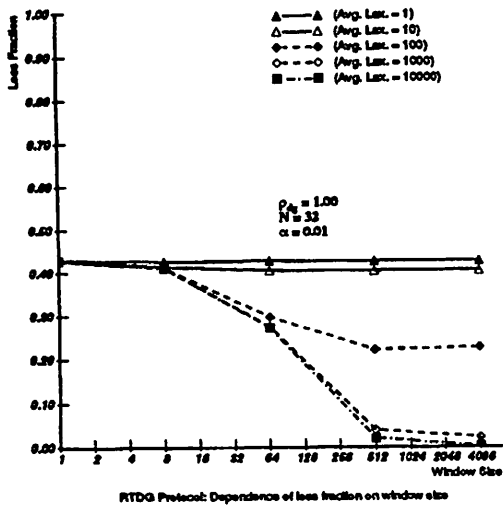
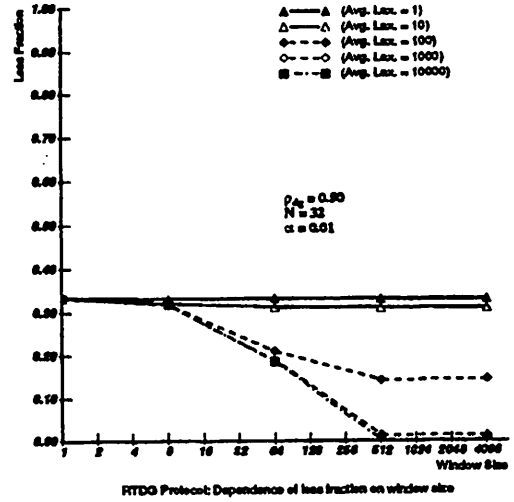
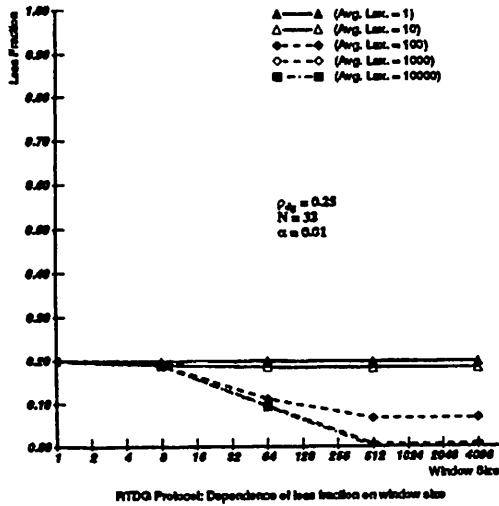


Figure 6.9 Dependence of loss fraction on window size ($N = 32$)

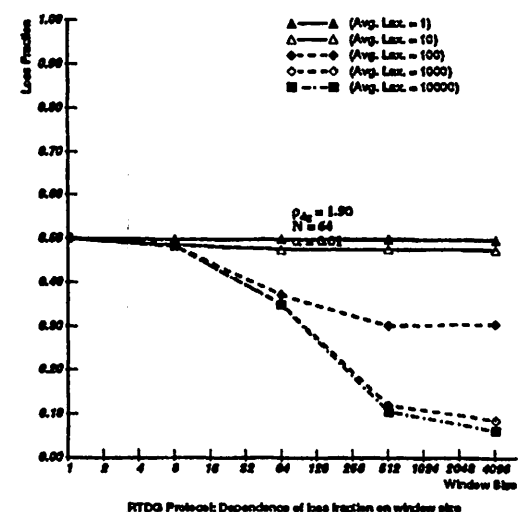
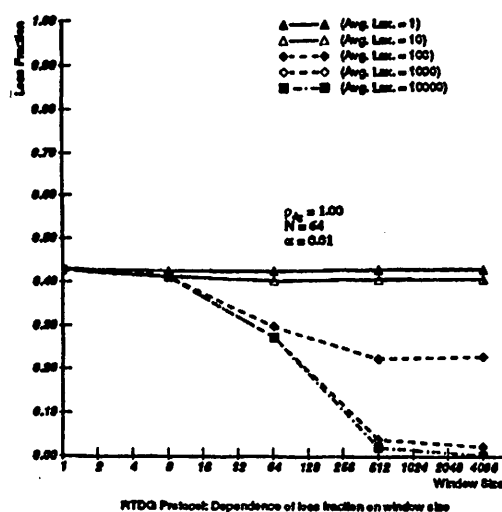
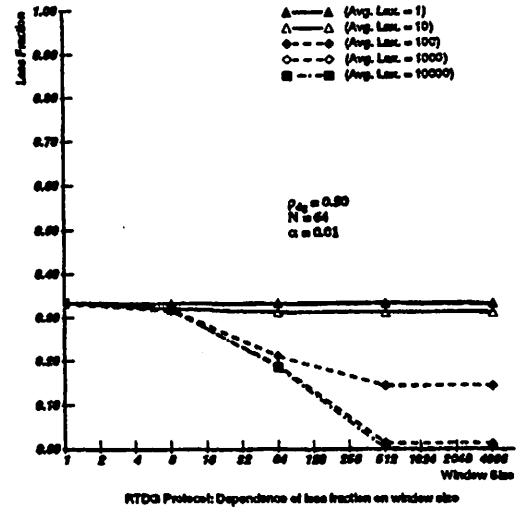
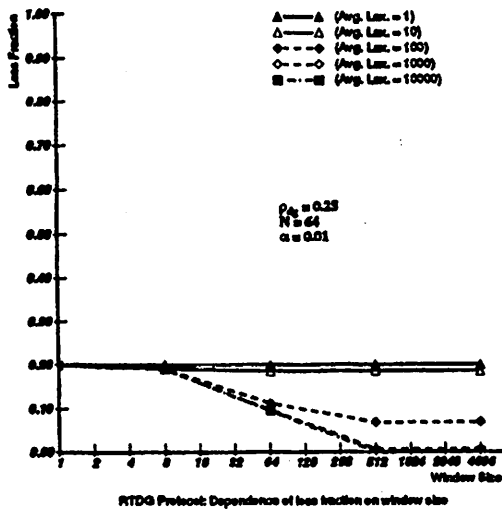


Figure 6.10 Dependence of loss fraction on window size ($N = 64$)

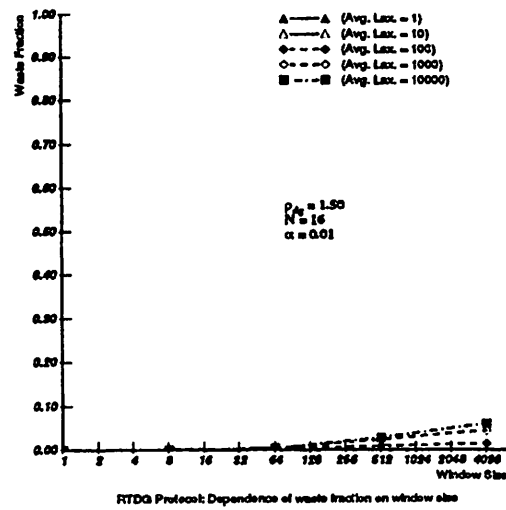
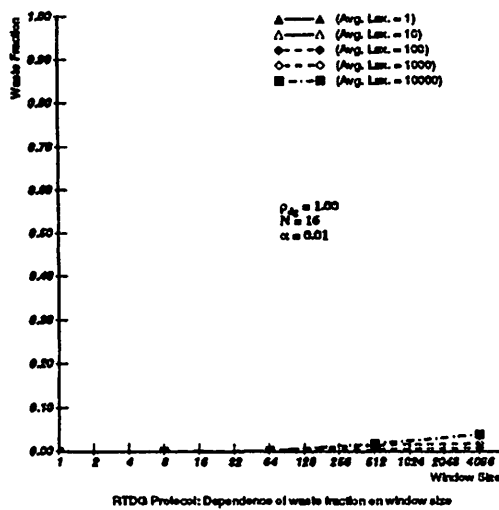
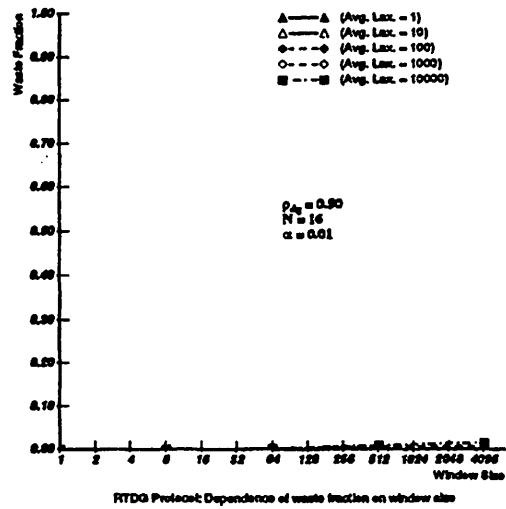
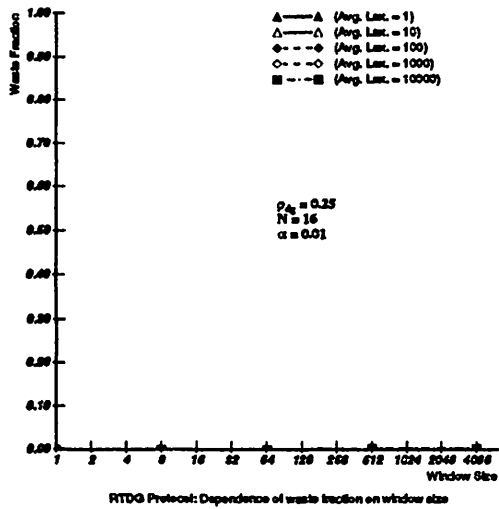


Figure 6.11 Dependence of waste fraction on window size ($N = 16$)

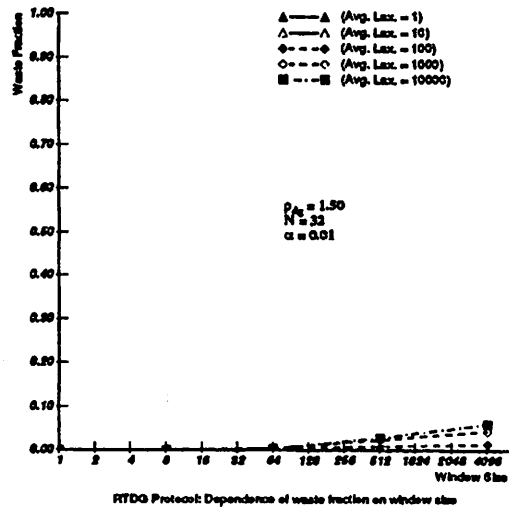
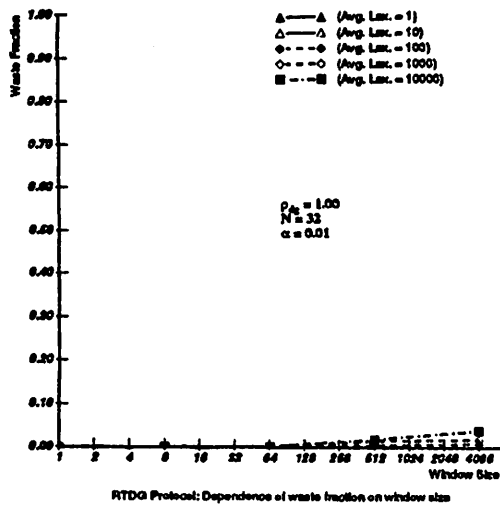
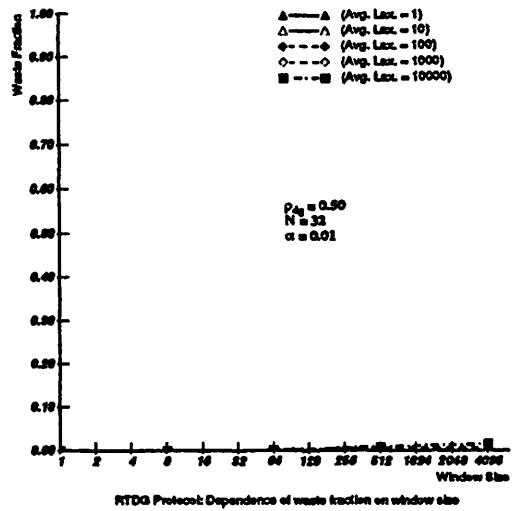
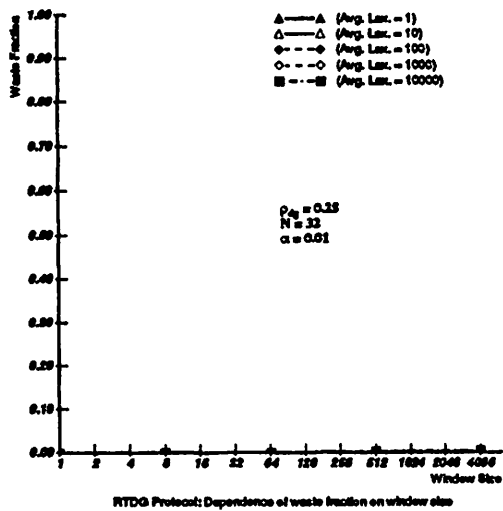


Figure 6.12 Dependence of waste fraction on window size ($N = 32$)

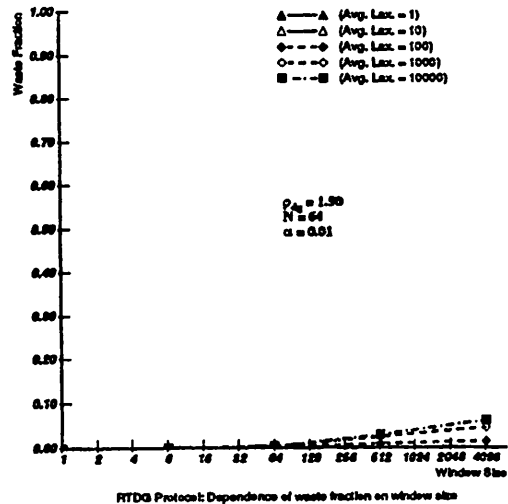
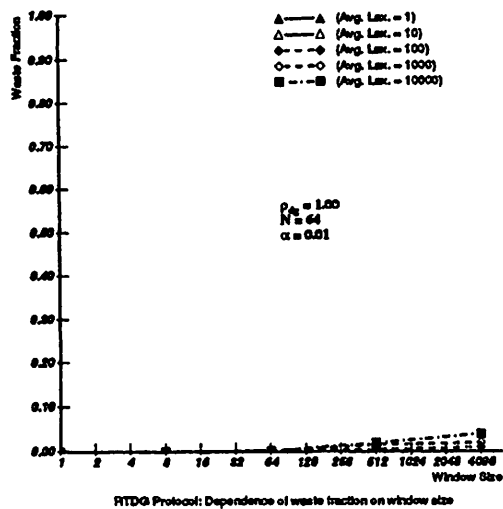
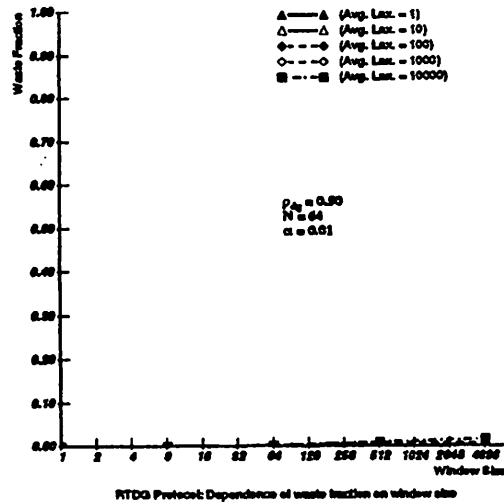
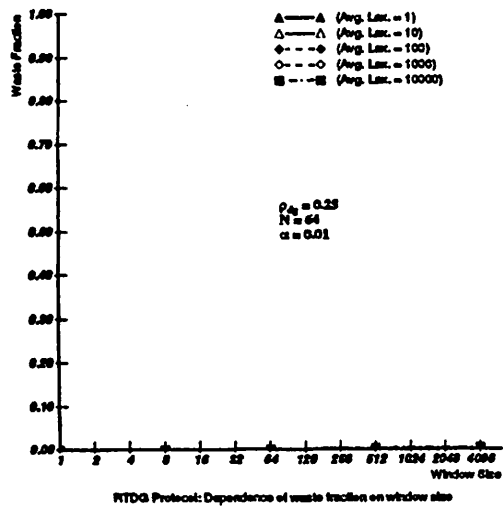


Figure 6.13 Dependence of waste fraction on window size ($N = 64$)

nodes (N). It may be seen that, in general, the waste fraction increases with the window size and is small in magnitude (e.g., it is about 4%, when the window size is 1024 and the load is 1.5).

Comparison of RTDG, CML and INRT

Figure 6.14 contains a comparison of the performance, measured in terms of the loss fraction, of the RTDG window protocol with the baseline CML and INRT protocols. The RTDG window protocol is clearly superior to the INRT protocol whose loss fraction represents a lower bound on the loss fraction achievable by any non-real-time protocol. The performance of the RTDG window protocol may also be observed to be very close to that of the CML protocol. Thus the RTDG window protocol closely implements the global minimum laxity first policy for channel arbitration.

6.4.4.2 RTVC Window Protocol

Dependence of Guarantee Ratio on Average Laxity

Figure 6.15 depicts the dependence of the guarantee ratio for the RTVC window protocol on the average laxity of real-time virtual circuit packets for two representative values of the number of real-time virtual circuits (N_{rtvc}) in the system. It may be seen that, in general, the guarantee ratio increases and approaches unity as the average laxity increases. This is to be expected, since packets with higher laxities are more likely to be guaranteed. The guarantee ratio may also be seen to decrease for a given average laxity, when the number of real-time virtual circuits is increased. This is because the worst case packet service time increases when the number of real-time virtual circuits increases. Hence fewer packets get guaranteed.

Note that, for any given number of real-time virtual circuits, there is a cutoff point (e.g., the point corresponding to an average laxity of 300 time units when $N_{rtvc} = 16$) up to which the guarantee ratio is zero. Such a cutoff point is to be expected, since the guarantee ratio is dependent on the worst case channel access

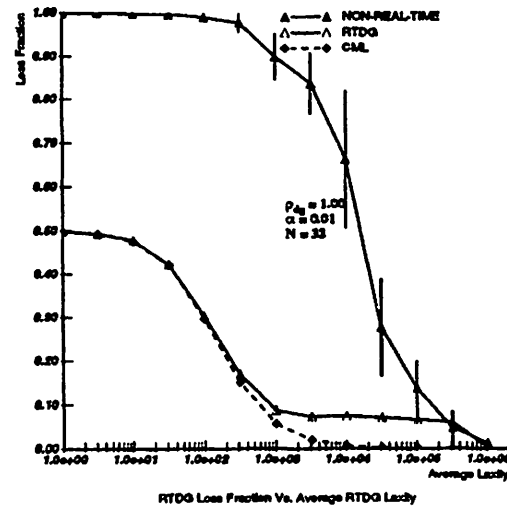
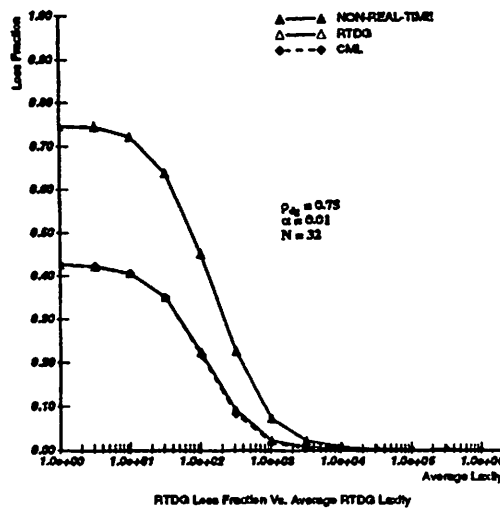
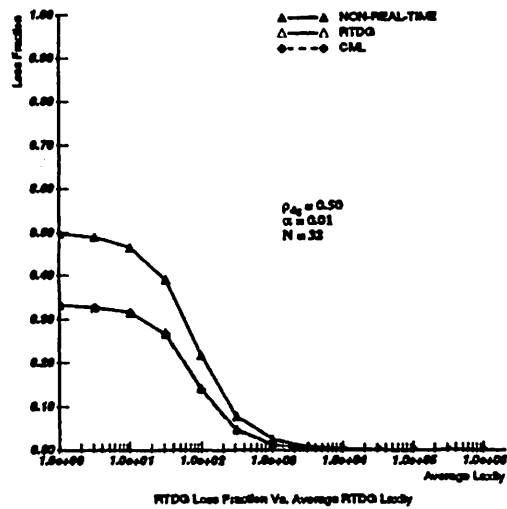
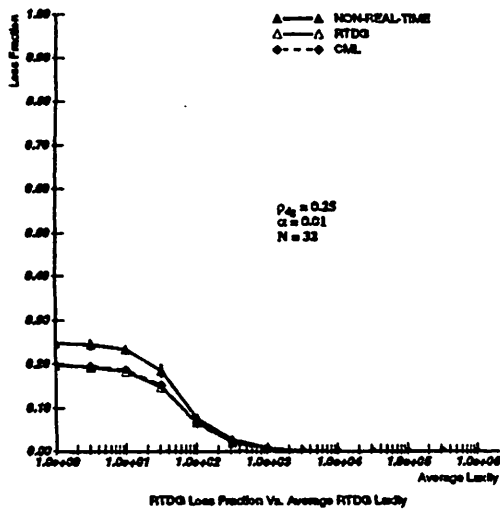


Figure 6.14 Comparison of RTDG, CML and INRT

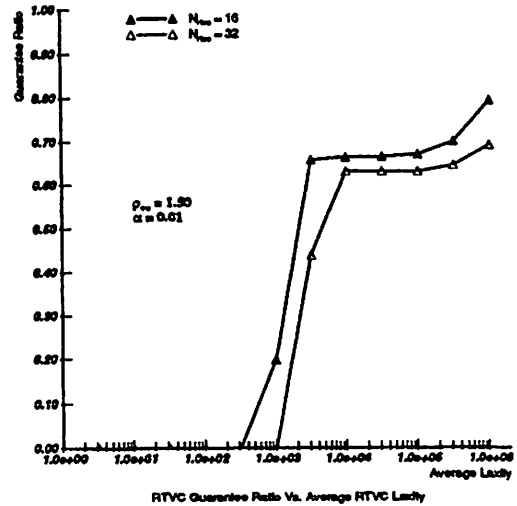
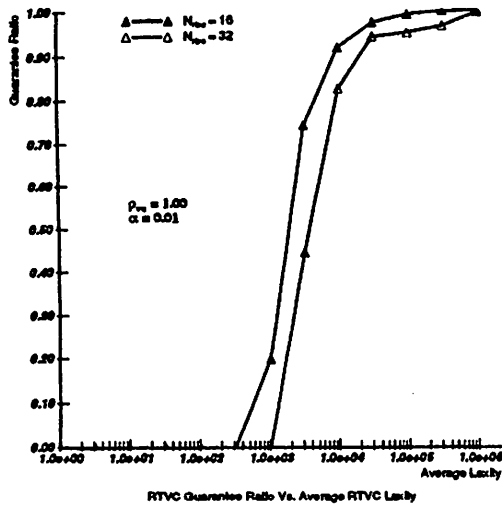
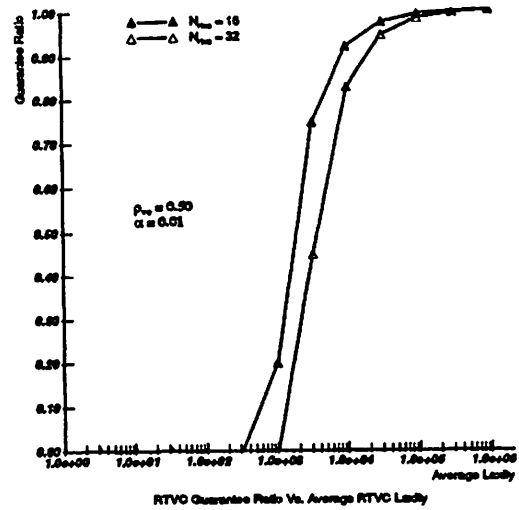
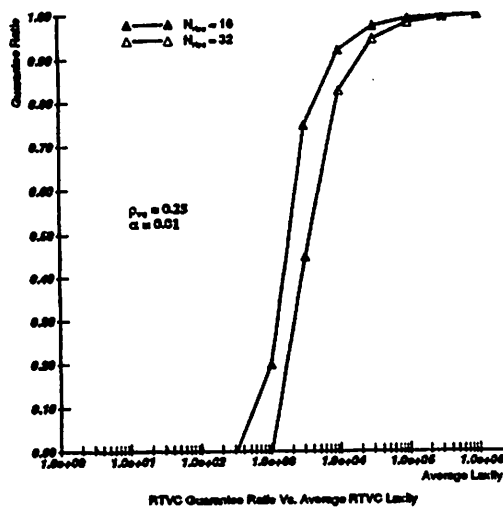


Figure 6.15 Dependence of guarantee ratio on average laxity

time for the packet at the head of a RTVC queue. If the laxity of an arriving packet is less than the worst case channel access time then the packet cannot be guaranteed, even if the queue in which it arrives is empty at the time of arrival. This cutoff point shifts to the right as the number of real-time virtual circuits increases.

Comparison of RTVC, INRT and TDMA

Figures 6.16 and 6.17 provide a comparison of the performance (measured in terms of the guarantee ratio) of the RTVC window protocol (for two representative values (16 and 32) of the number of real-time virtual circuits (N_{rtvc}) in the system) with the baseline IRTVC protocol and the TDMA protocol. It may be observed that the performance of the RTVC window protocol is close to that of the idealized baseline protocol. Thus, if some other protocol (e.g., a token passing protocol) can provide a better performance than that provided by the RTVC window protocol, then the improvement in performance achievable by using the other protocol is not significant. It may be noted that the performance of the RTVC window protocol is better than that of the TDMA protocol under certain conditions.

6.4.4.3 INTPVC Window Protocol

Effect of Integration on Guarantee Ratio

Figures 6.18 and 6.19 provide a comparison (measured in terms of the guarantee ratio), of the performance of the INTPVC window protocol with the non-integrated RTVC protocol, for two representative values (0.5 and 1) of the real-time datagram load (ρ_{dg}) and two representative values (16 and 32) of the number of real-time virtual circuits (N_{rtvc}) in the system. It may be observed that the performance of the INTPVC window protocol is better than that of the RTVC window protocol. Thus use of the *integrated* protocol INTPVC has had the effect of improving the guarantee ratio.

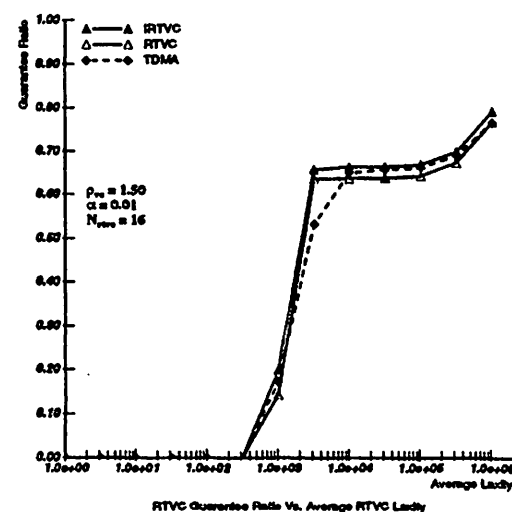
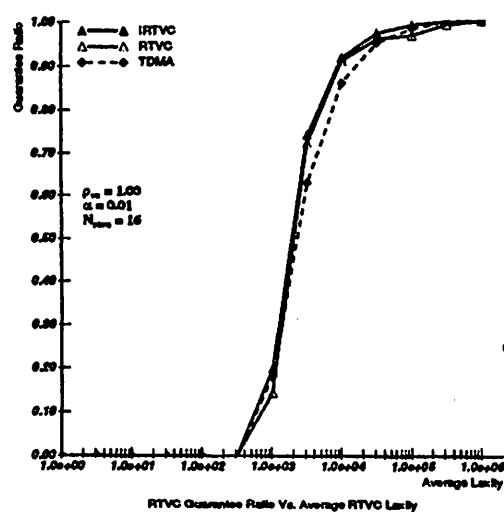
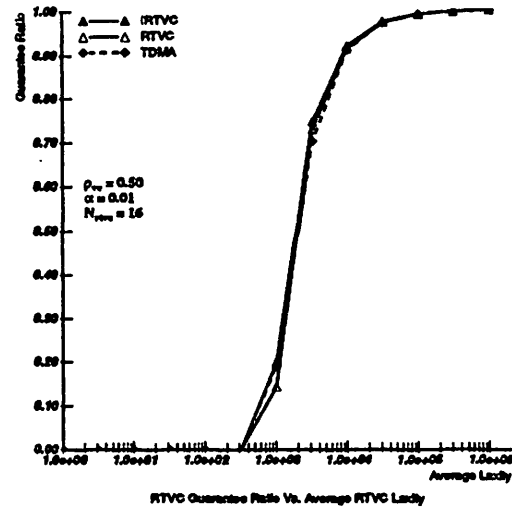
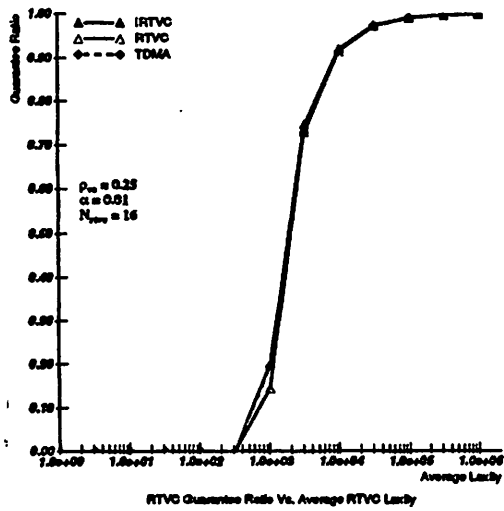


Figure 6.16 Comparison of RTVC, IRTVC and TDMA ($N_{rtvc} = 16$)

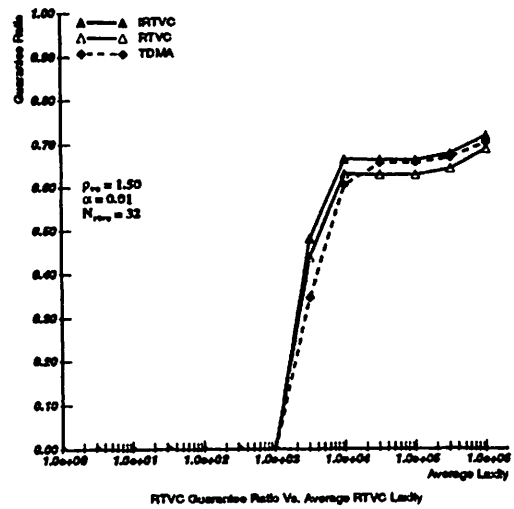
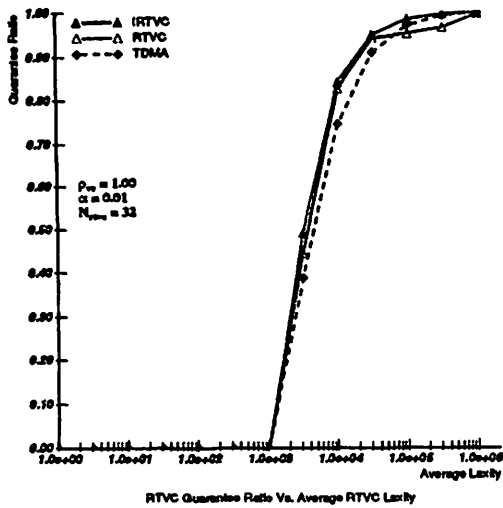
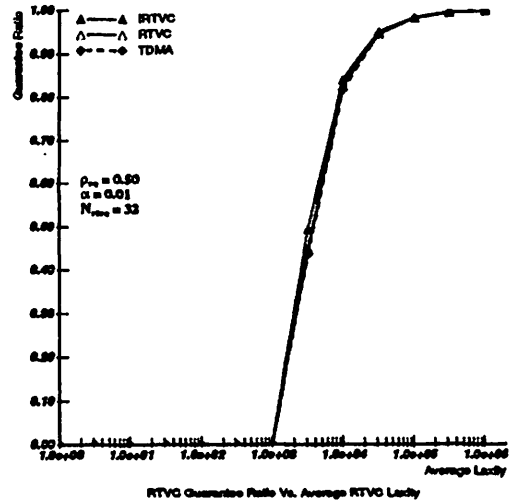
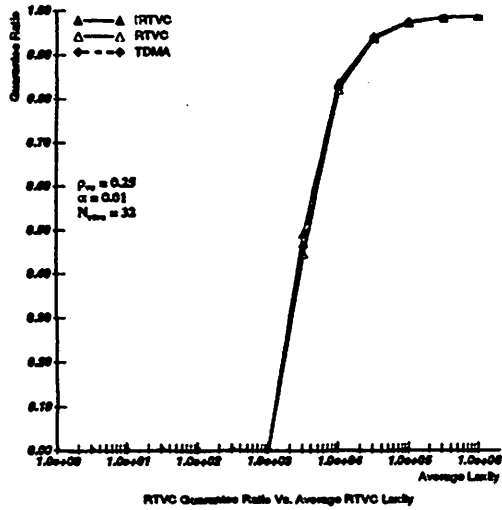


Figure 6.17 Comparison of RTVC, IRTVC and TDMA ($N_{rsvc} = 32$)

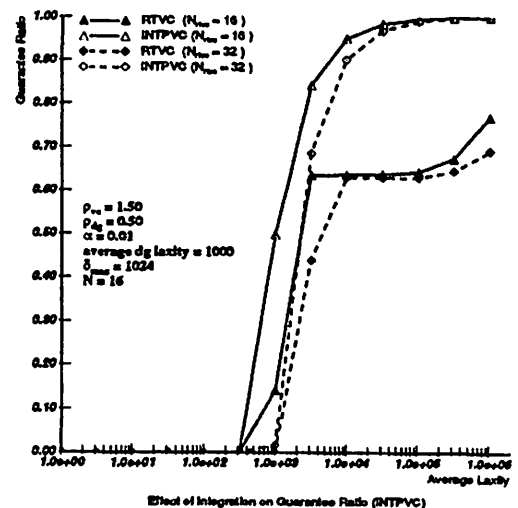
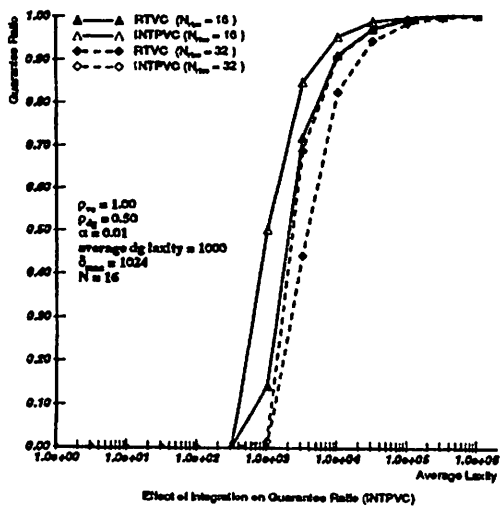
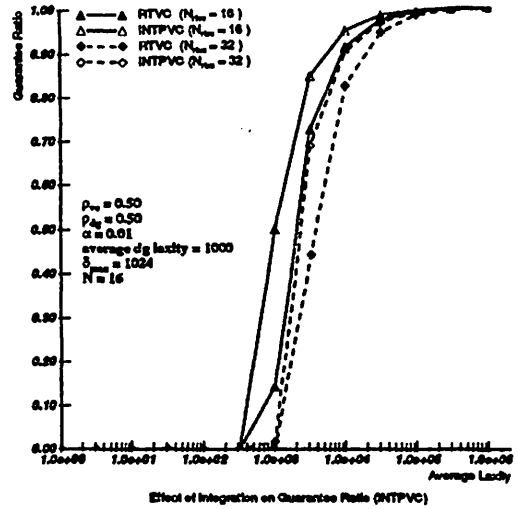
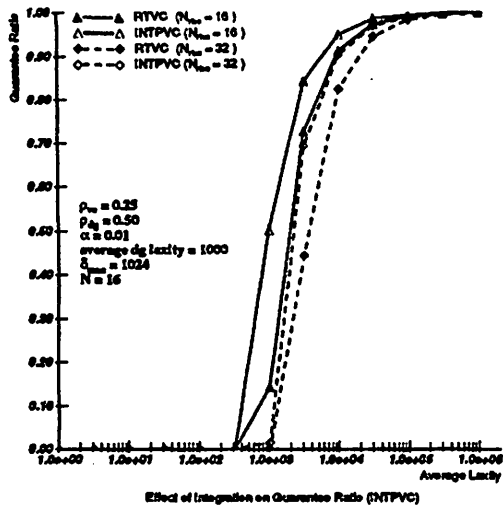


Figure 6.18 Effect of integration on guarantee ratio ($\rho_{dg} = 0.50$)

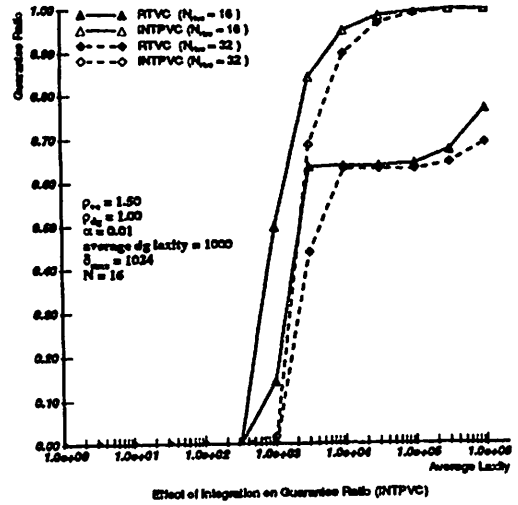
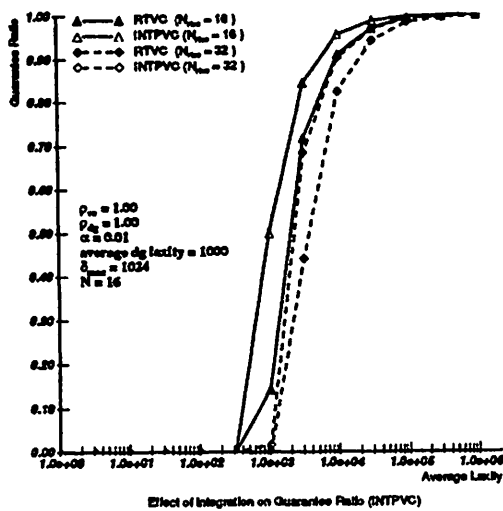
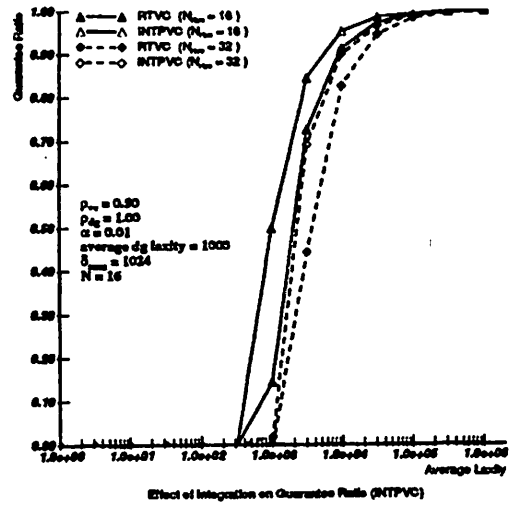
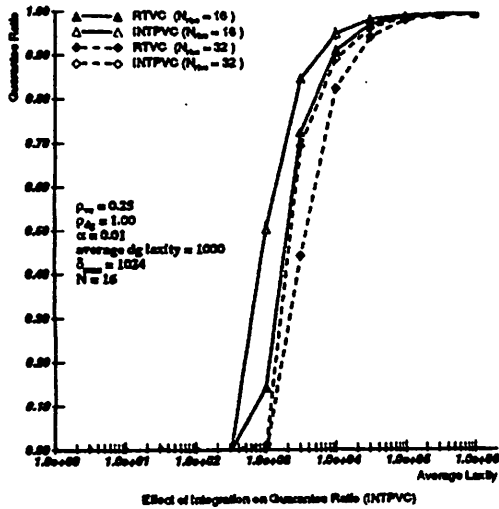


Figure 6.19 Effect of integration on guarantee ratio ($\rho_{dg} = 1.00$)

Comparison of Guarantee Ratio for VCTIMER and INTPVC

Figures 6.20 and 6.21 provide a comparison of the performance, measured in terms of the guarantee ratio, of the INTPVC window protocol with the baseline VCTIMER protocol. It may be observed that the performance of the INTPVC window protocol is better than that of the baseline protocol. This is so because the worst case channel access time for the INTPVC protocol is smaller than that for the VCTIMER baseline protocol.

Effect of Integration on Loss Fraction

Figures 6.22 and 6.23 provide a comparison of the performance (measured in terms of the loss fraction) of the INTPVC window protocol with the non-integrated RTDG protocol, for two representative values (0.5 and 1.0) of the real-time datagram load (ρ_{dg}) and two representative values (16 and 32) of the number of real-time virtual circuits. It may be observed that the performance of the INTPVC window protocol is better (except at very low laxities of the order of the contention resolution overhead) than that of the RTDG window protocol up to a certain real-time virtual circuit load (which depends on the real-time datagram load). Beyond this real-time virtual circuit load, the performance of the integrated protocol is poorer than that of the non-integrated protocol. For example, in Figure 6.22, the performance of the integrated protocol is better than that of RTDG when the average laxity is greater than 10, $\rho_{vc} = 0.25$ and $N_{rtvc} = 16$. However the difference in performance decreases as the real-time virtual circuit load increases. When the real-time virtual circuit load is 1.5, the loss fraction for INTPVC is higher than that for RTDG. Thus integration has had the effect of improving the loss fraction for the INTPVC protocol over a range of real-time virtual circuit loads. Beyond this range, the improvement in the quality of servicing real-time virtual circuit packets is maintained (as was seen earlier), but at the expense of the quality of service for the real-time datagrams.

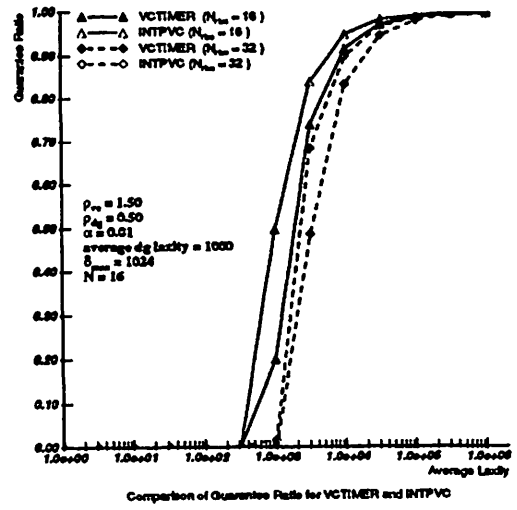
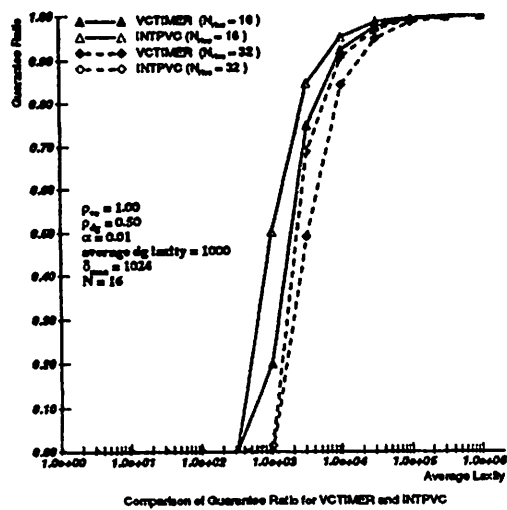
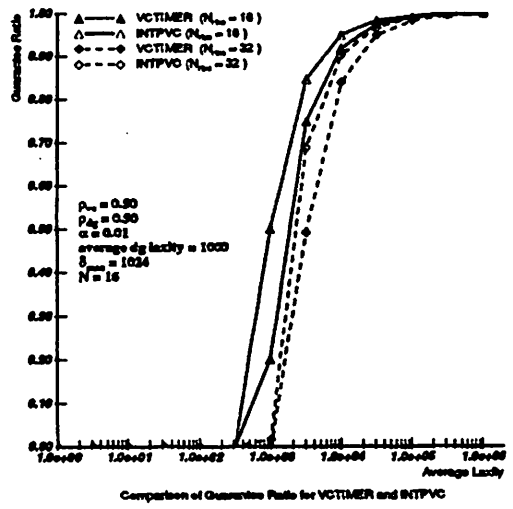
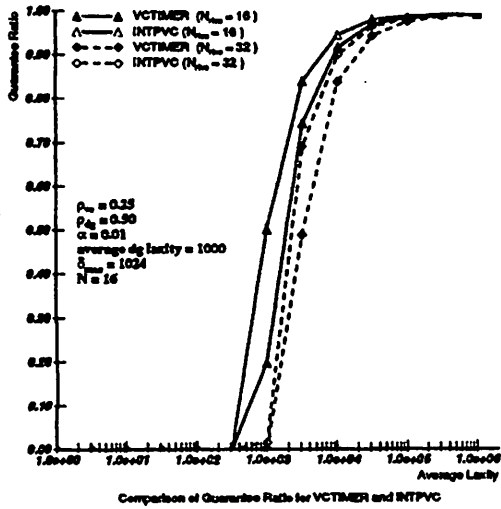


Figure 6.20 Guarantee ratio: VCTIMER vs. INTPVC ($\rho_{dg} = 0.50$)

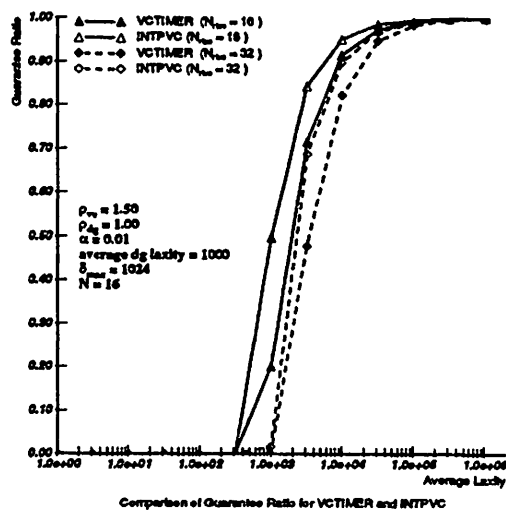
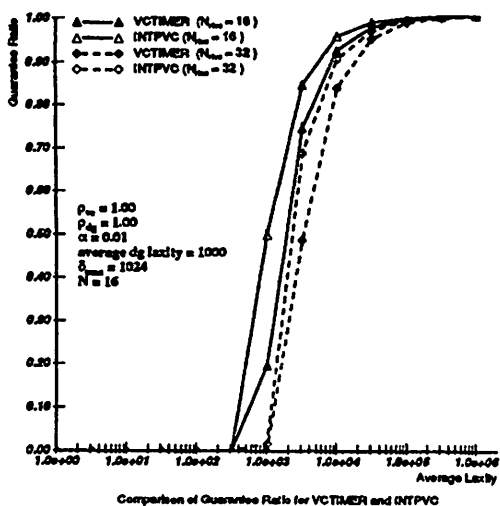
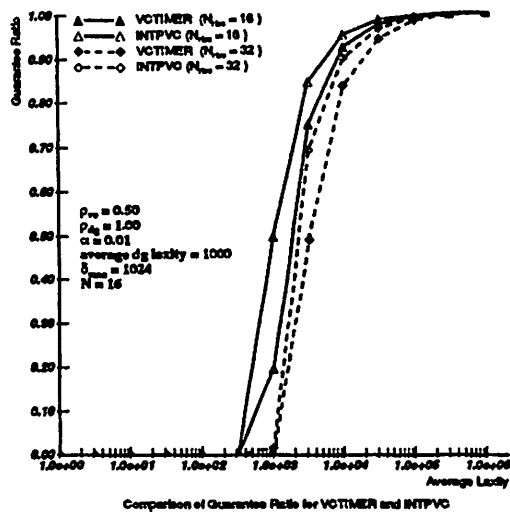
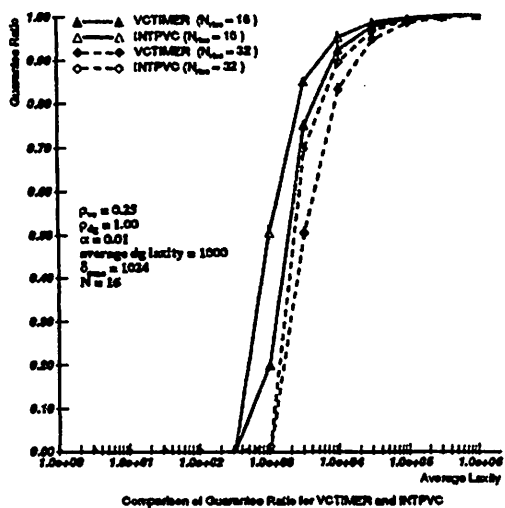


Figure 6.21 Guarantee ratio: VCTIMER vs. INTPVC ($\rho_{dg} = 1.00$)

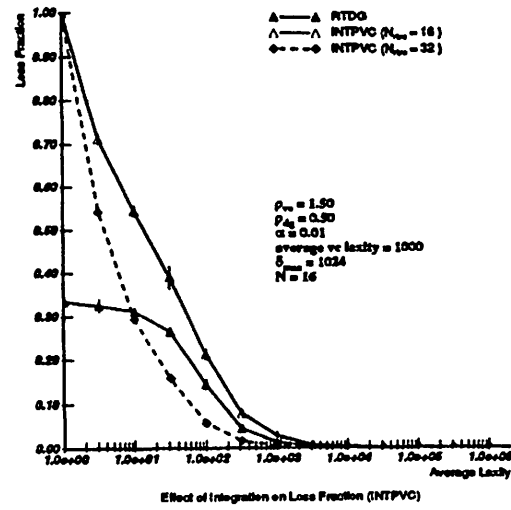
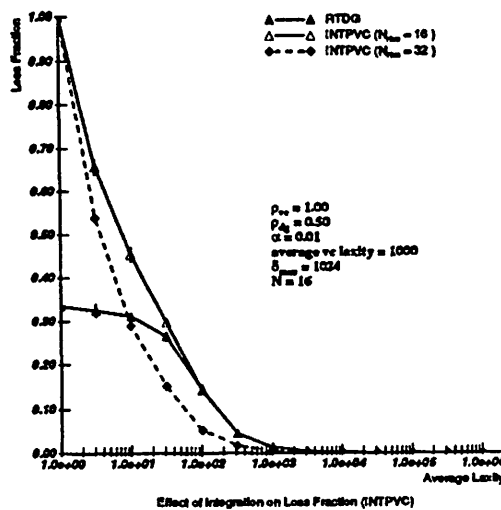
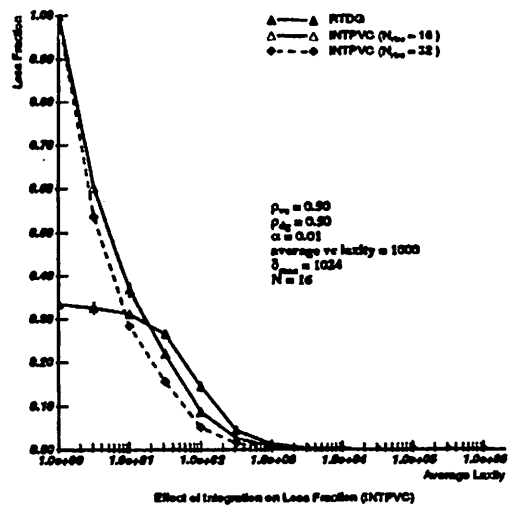
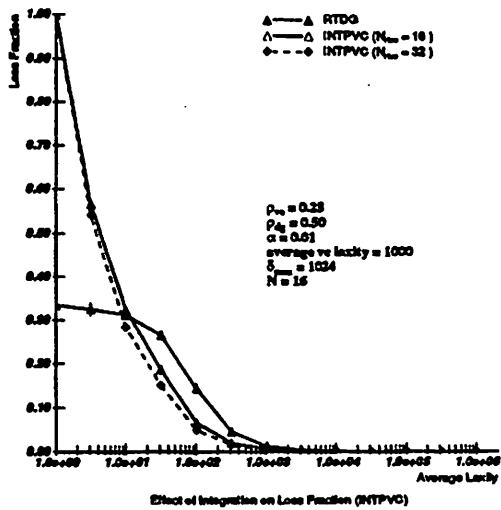


Figure 6.22 Effect of integration on loss fraction ($\rho_{dg} = 0.50$)

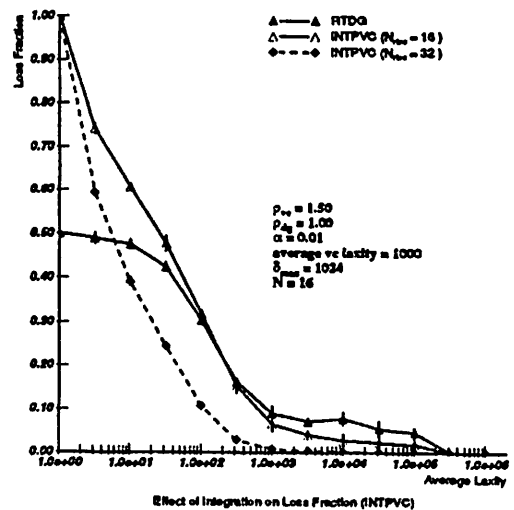
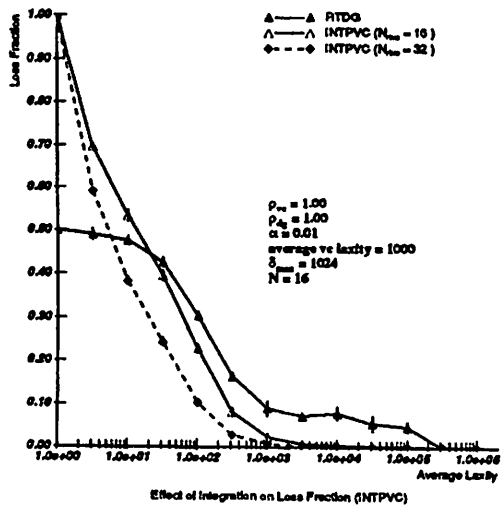
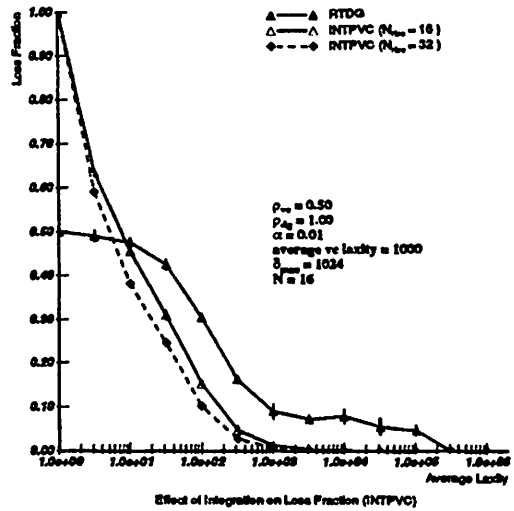
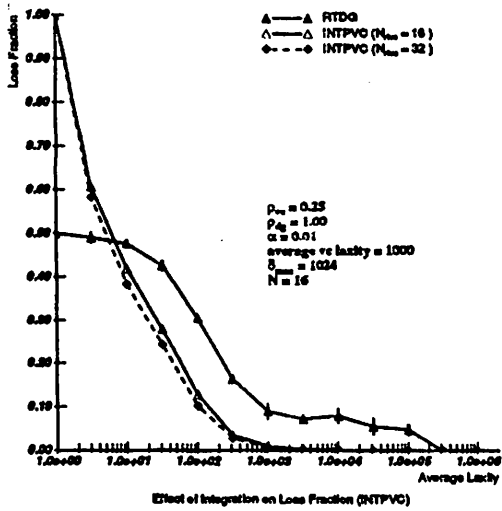


Figure 6.23 Effect of integration on loss fraction ($\rho_{dg} = 1.00$)

Comparison of Loss Fraction for VCTIMER and INTPVC

Figures 6.24 and 6.25 provide a comparison of the performance (measured in terms of the loss fraction) of the INTPVC window protocol (which we noted is biased towards real-time virtual circuit packets) with the baseline VCTIMER protocol. It may be observed that the loss fraction for the INTPVC window protocol is higher than that for the baseline VCTIMER protocol at low real-time datagram loads ($\rho_{dg} = 0.5$) in Figure 6.24. However, as the real-time datagram load is increased (Figure 6.25), the difference in performance between the INTPVC window protocol and the baseline VCTIMER protocol decreases and may even become negative.

6.4.4.4 INTPDG Window Protocol

Effect of Integration on Guarantee Ratio

Figures 6.26 and 6.27 provide a comparison of the performance of the INTPDG window protocol with the non-integrated RTVC protocol, measured in terms of the guarantee ratio. It may be observed that the performance of the INTPDG window protocol (which is biased to servicing real-time datagrams) is poorer than that of the RTVC window protocol. Thus integration using INTPDG has had the effect of reducing the quality of performance for real-time virtual circuit packets.

Comparison of Guarantee Ratio for VCTIMER and INTPDG

Figures 6.28 and 6.29 provide a comparison of the performance of the INTPDG window protocol (which we noted is biased towards real-time datagrams) with the baseline VCTIMER protocol, measured in terms of the guarantee ratio. It may be observed that, in general, the guarantee ratio for the INTPDG window protocol is lower than that for the baseline VCTIMER protocol.

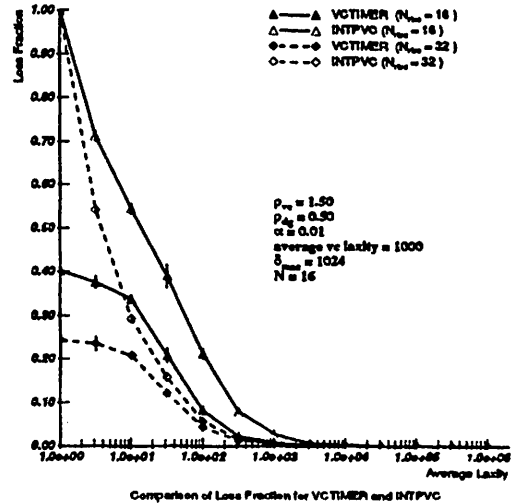
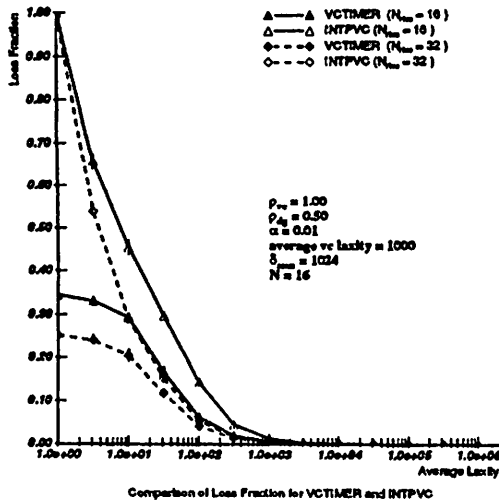
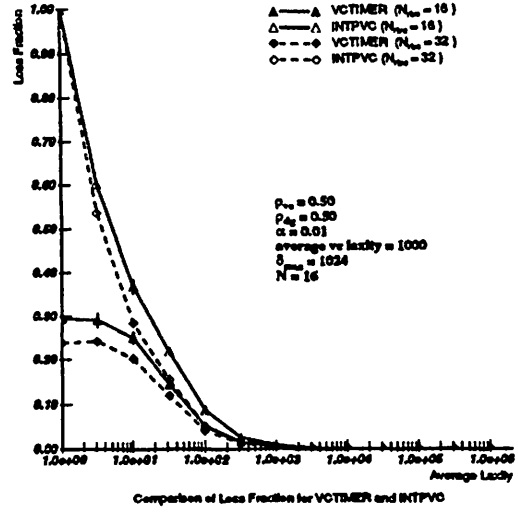
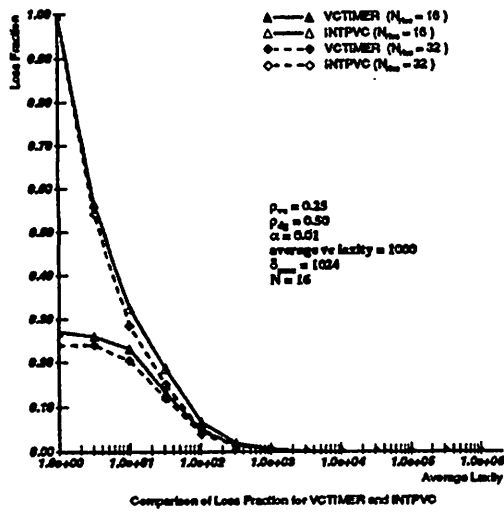


Figure 6.24 Loss fraction: VCTIMER vs. INTPVC ($\rho_{dg} = 0.50$)

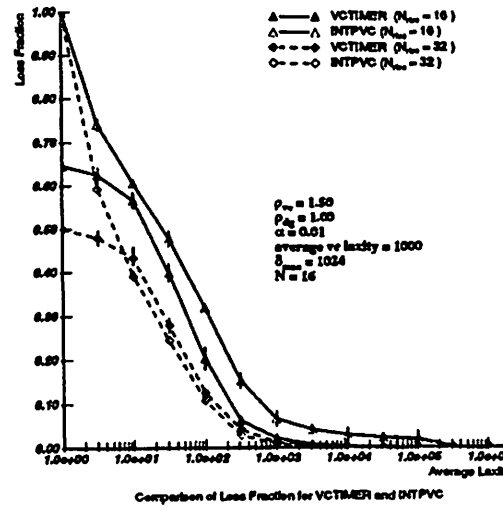
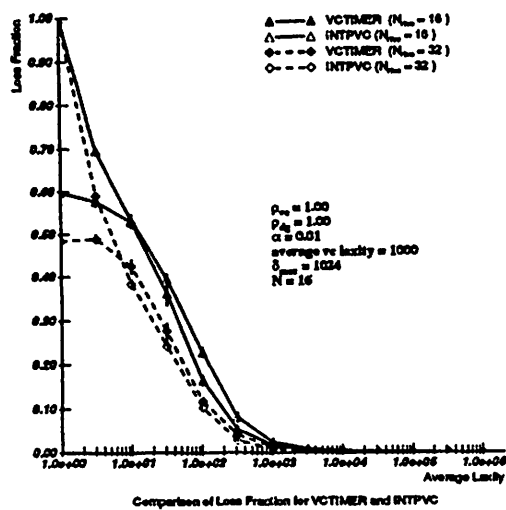
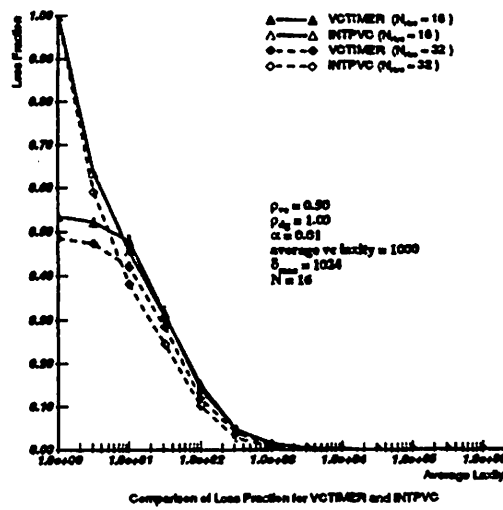
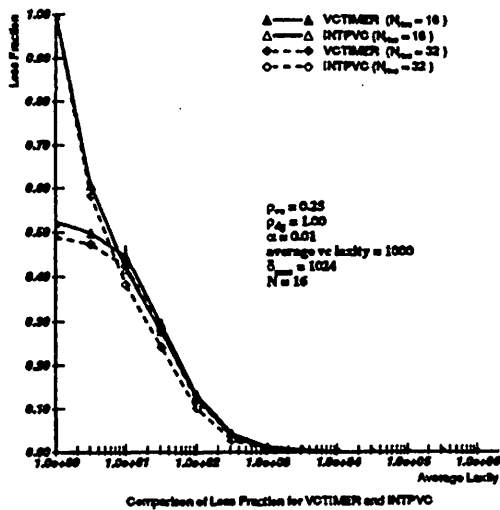


Figure 6.25 Loss fraction: VCTIMER vs. INTPVC ($\rho_{dg} = 1.00$)

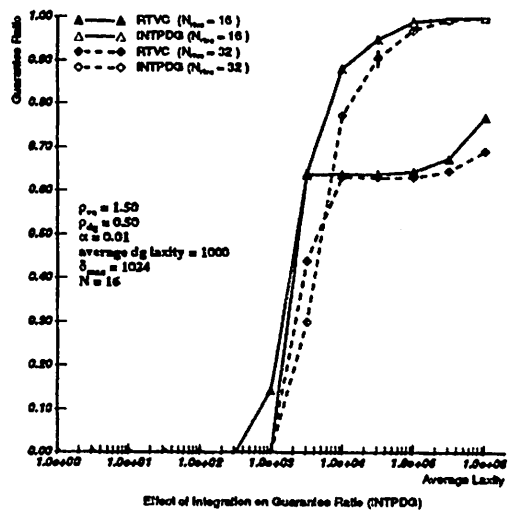
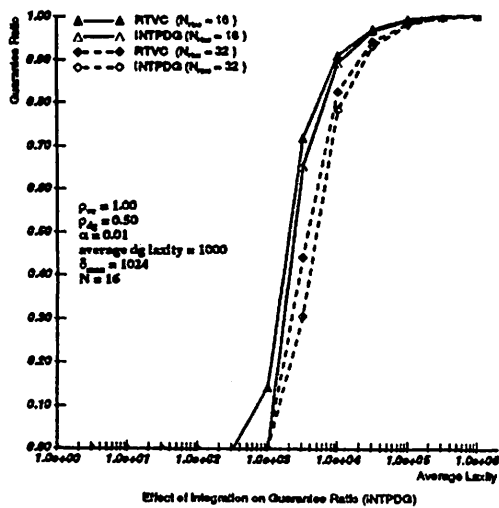
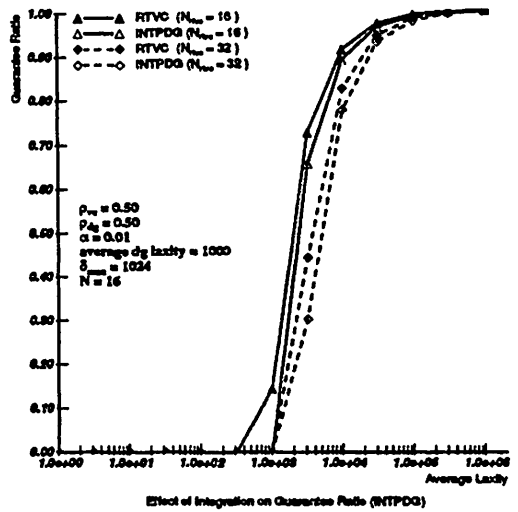
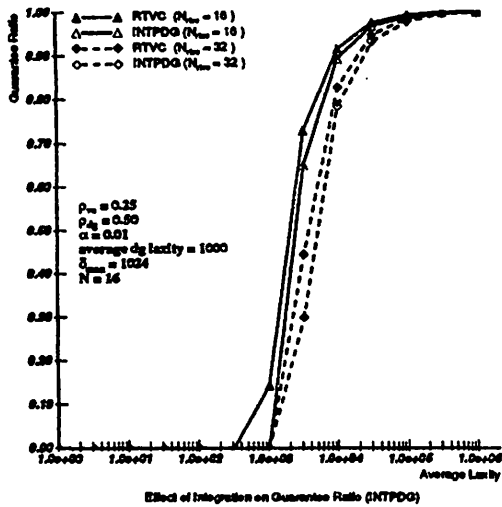


Figure 6.26 Effect of integration on INTPDG guarantee ratio ($\rho_{dg} = 0.50$)

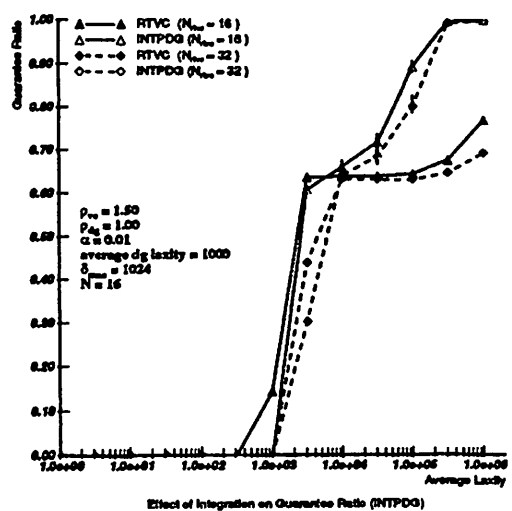
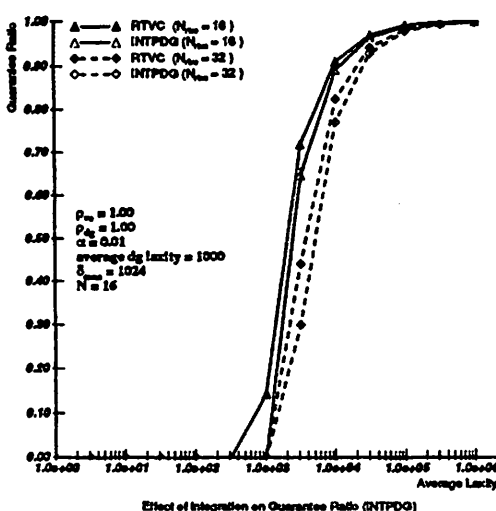
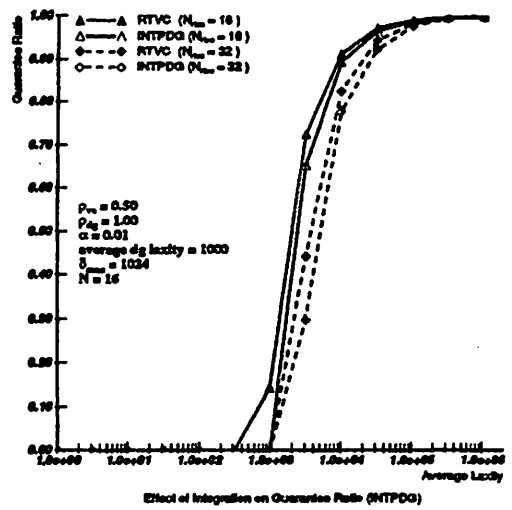
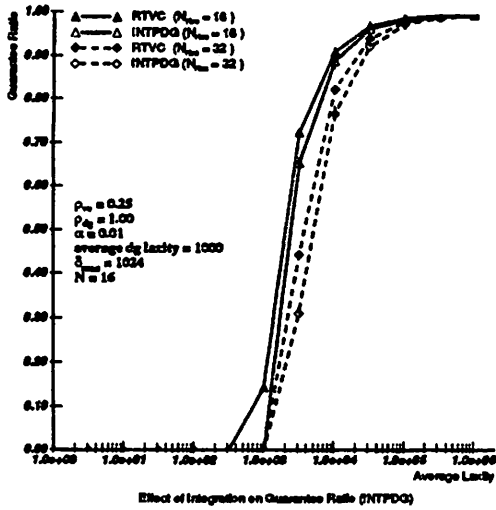


Figure 6.27 Effect of integration on INTPDG guarantee ratio ($\rho_{dg} = 1.00$)

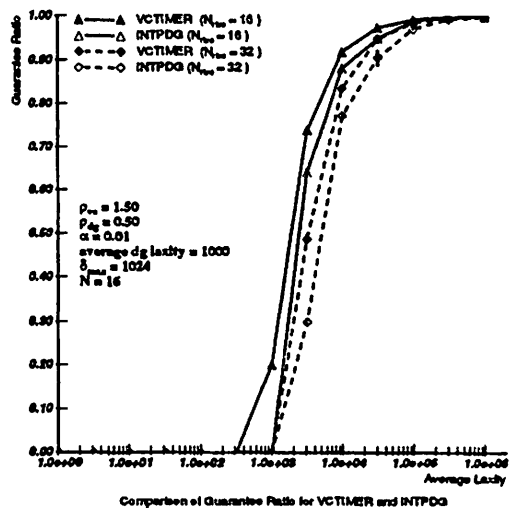
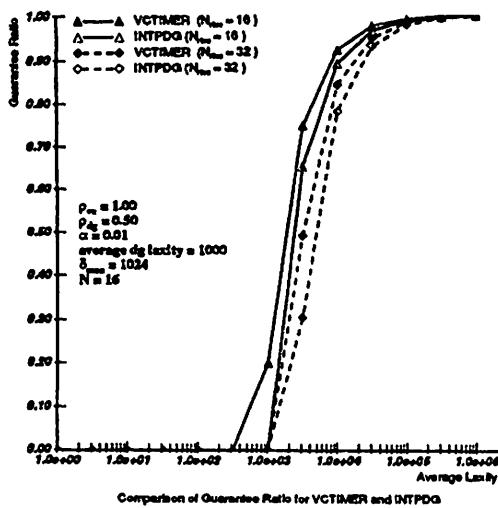
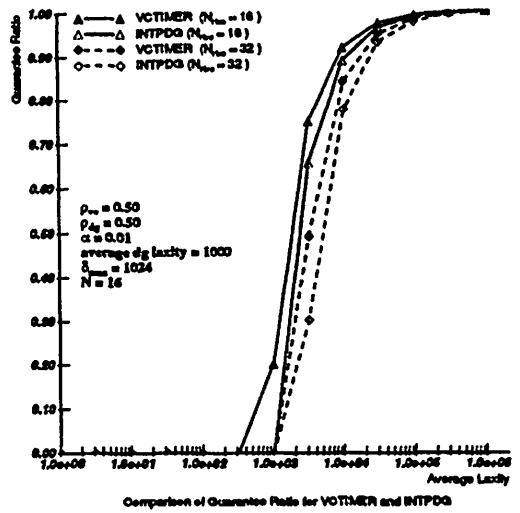
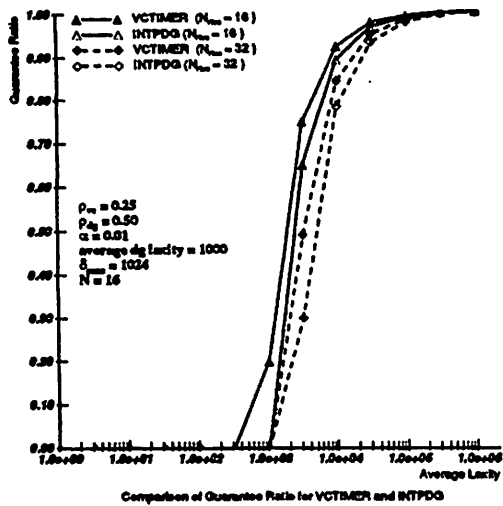


Figure 6.28 Guarantee ratio: VCTIMER vs. INTPDG ($\rho_{dg} = 0.50$)

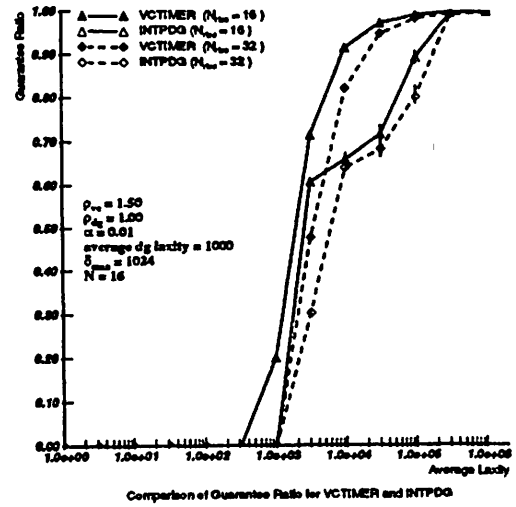
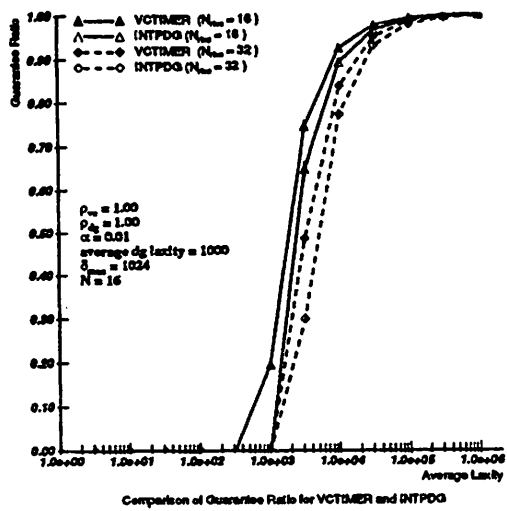
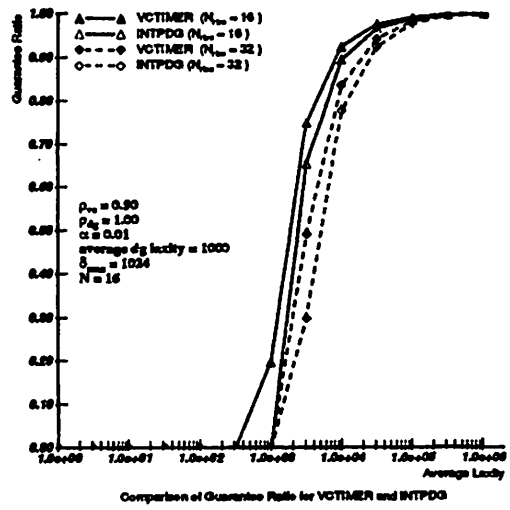
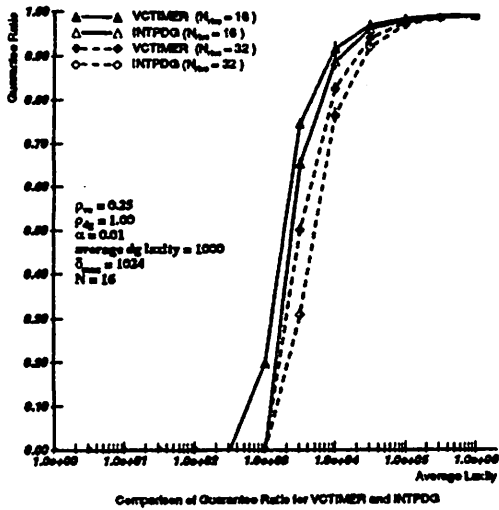


Figure 6.29 Guarantee ratio: VCTIMER vs. INTPDG ($\rho_{dg} = 1.00$)

Effect of Integration on Loss Fraction

Figures 6.30 and 6.31 provide a comparison of the performance of the INTPDG window protocol with the non-integrated RTDG protocol, measured in terms of the loss fraction. It may be observed that the performance of the INTPDG window protocol is better (except at very low laxities of the order of the contention resolution overhead) than that of the RTDG window protocol. Thus integration has had the effect of improving the loss fraction for the INTPDG protocol.

Comparison of Loss Fraction for VCTIMER and INTPDG

Figures 6.32 and 6.33 provide a comparison of the performance of the INTPDG window protocol with the baseline VCTIMER protocol, measured in terms of the loss fraction. It may be observed that, in general, the loss fraction for the INTPDG window protocol is smaller than that for the baseline VCTIMER protocol (especially at higher datagram loads - Figure 6.33). However at smaller real-time datagram loads (Figure 6.32) the difference in performance is smaller, except at low laxities of the order of the contention resolution overhead, when INTPDG has a higher loss fraction than VCTIMER.

6.5 Summary

In this chapter, we presented the results of a simulation study conducted in order to evaluate the performance of the window MAC protocols developed in the previous chapters. We defined the performance metrics, viz., *loss fraction* and *guarantee ratio* that are appropriate for evaluating the MAC protocols that support the implementation of RTCLS and RTCOS respectively. We defined four idealized baseline protocols, viz., CML, INRT, IRTVC and VCTIMER that we used as baseline protocols in the evaluation of the window protocols. We described the system model assumed by the simulation programs. Finally we presented the results of the simulation study. The results indicate that the protocol RTDG closely approximates the idealized baseline protocol CML. The performance of RTDG is also superior to that

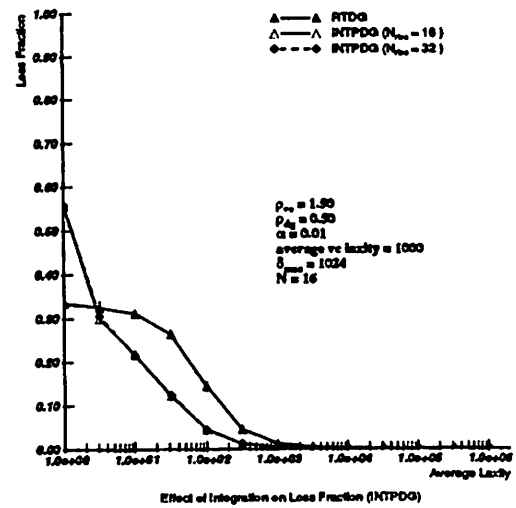
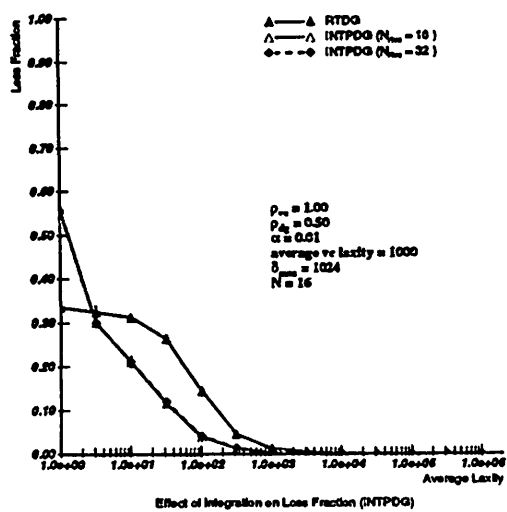
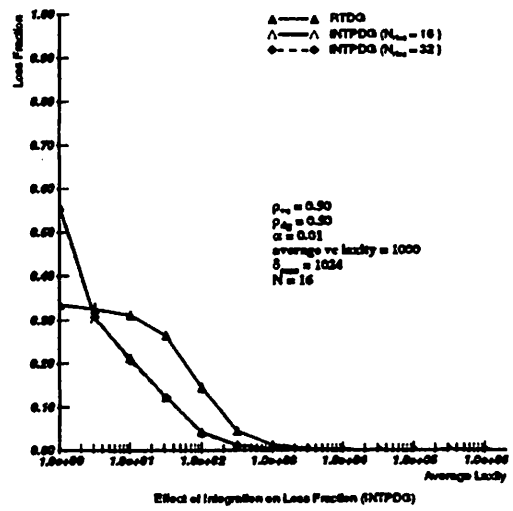
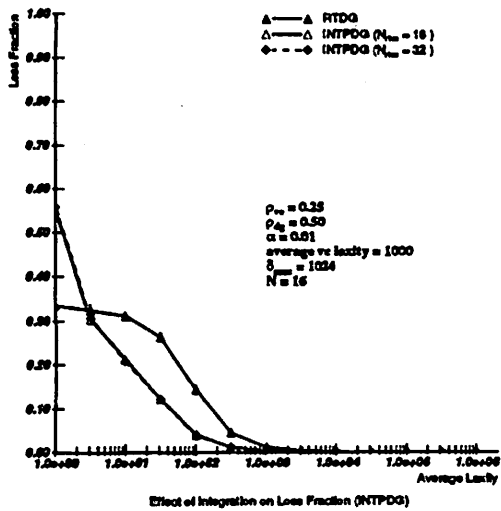


Figure 6.30 Effect of integration on INTPDG loss fraction ($\rho_{dg} = 0.50$)

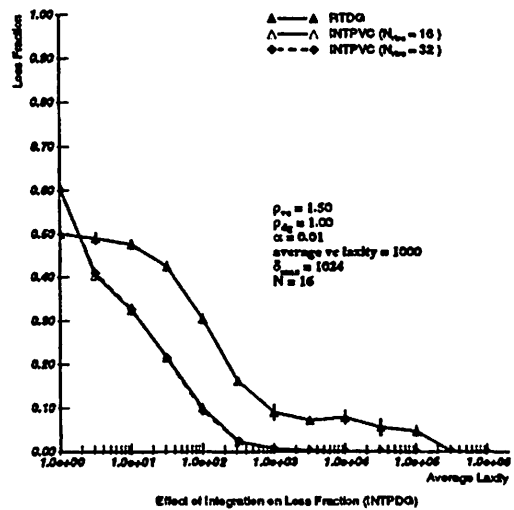
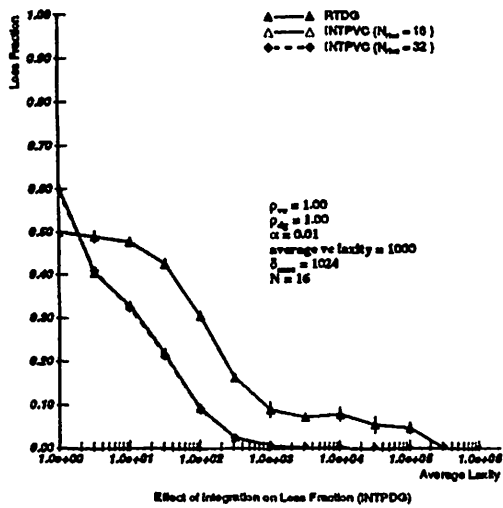
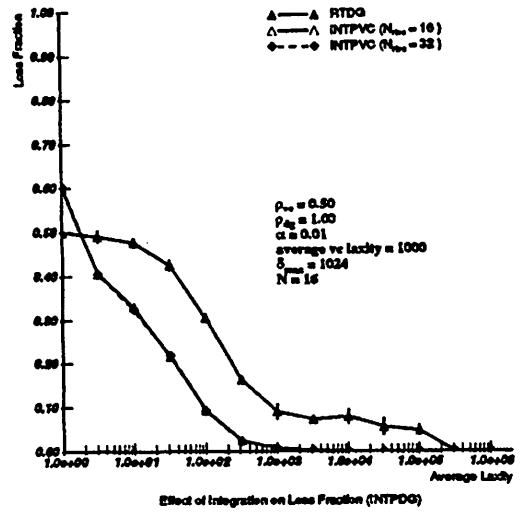
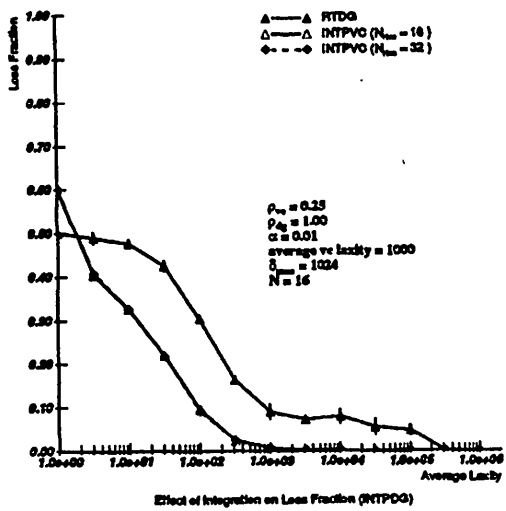
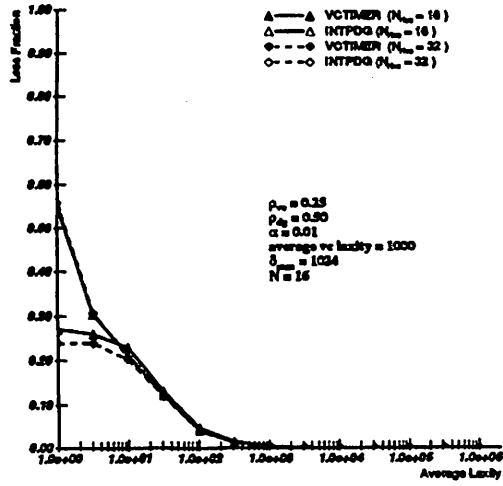
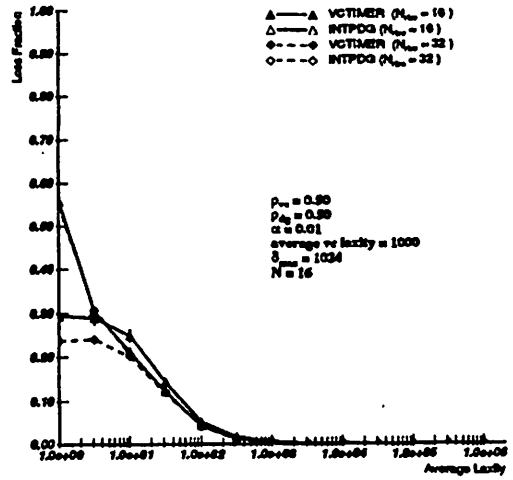


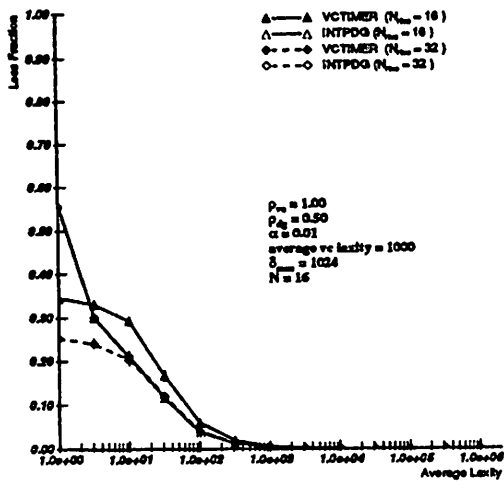
Figure 6.31 Effect of integration on INTPDG loss fraction ($\rho_{dg} = 1.00$)



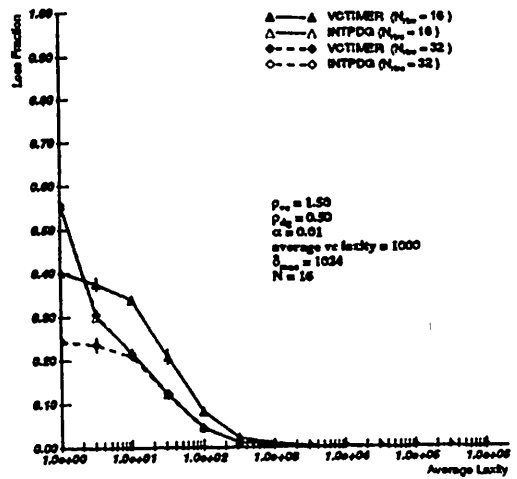
Comparison of Loss Fraction for VCTIMER and INTPDG



Comparison of Loss Fraction for VCTIMER and INTPDG



Comparison of Loss Fraction for VCTIMER and INTPDG



Comparison of Loss Fraction for VCTIMER and INTPDG

Figure 6.32 Loss fraction: VCTIMER vs. INTPDG ($\rho_{dg} = 0.50$)

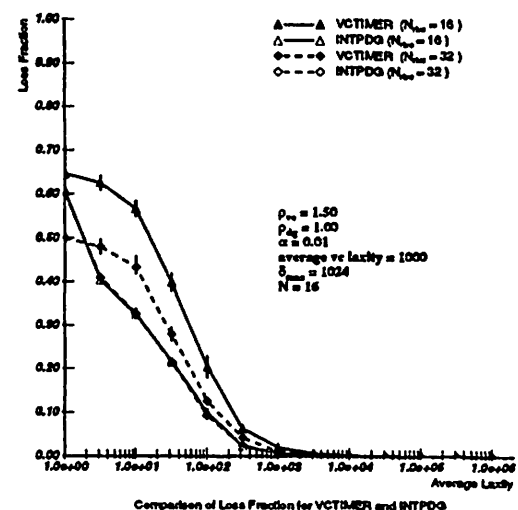
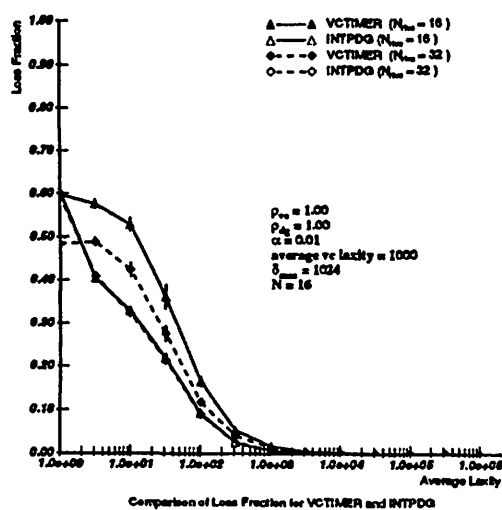
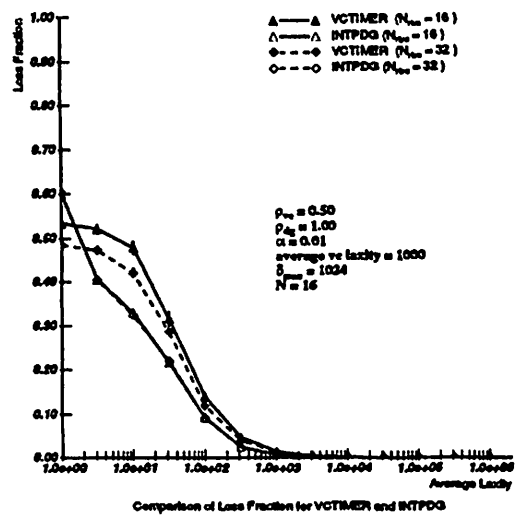
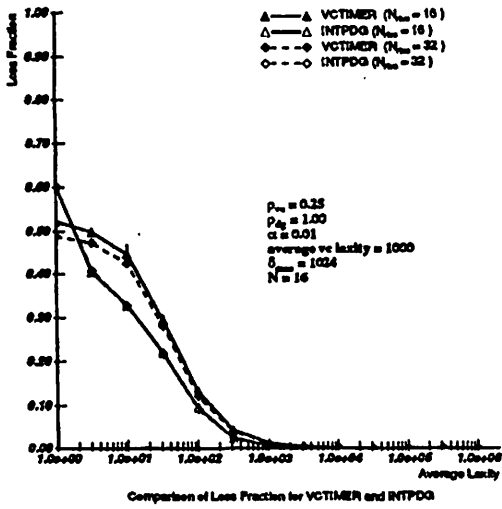


Figure 6.33 Loss fraction: VCTIMER vs. INTPDG ($\rho_{dg} = 1.00$)

of existing standards that have no notion of packet timing constraints. The protocol RTVC was shown to closely approximate the idealized baseline protocol IRTVC that also implements real-time virtual circuit arbitration. The INTPVC window protocol provides a better quality of service for real-time virtual circuit packets than the idealized baseline protocol VCTIMER. The INTPDG window protocol provides a better quality of service for real-time datagram packets than the VCTIMER baseline protocol.

CHAPTER 7

RESERVATION-BASED SCHEMES

7.1 Introduction

We introduced the notion of a real-time virtual circuit in Chapter 4. We defined a real-time virtual circuit as a logical channel with a bounded packet service time. We examined both collision free (e.g., TDMA, token-passing) and collision-based (window protocols) approaches to implementing real-time virtual circuits. Both classes of protocols have some form of overhead associated with them (e.g., the token passing delay in token passing protocols, the delay awaiting one's turn to transmit in TDMA and the wasted time due to collisions in the window protocol) and consequently the worst case channel access time for both classes of protocols is of approximately the same magnitude and roughly equal to the number of real-time virtual circuits times the packet transmission time. As a result, the difference in performance between the two classes of protocols in terms of the performance measure of interest, viz., *guarantee ratio*, is not substantial. This is because both classes of protocols always make use of the worst case channel access time in determining whether a packet that has just arrived can be guaranteed or not; i.e., they assume that all the other nodes will be constantly busy and hence a node will be able to transmit only one packet every cycle. Thus these are *pessimistic* protocols which can reject messages that may actually be able to make their deadlines. The pessimistic nature of these protocols arises because of two reasons:

1. The notion of *guarantee*: These protocols are expected to support the notion of guarantee, which means that they can accept a packet for transmission only if they are *absolutely* certain that the packet can be transmitted before its deadline.

2. *Local view*: These protocols lack global knowledge, i.e., each station knows only the state of its own transmission queues when it guarantees a message. Hence it is forced to assume that other stations will always have a packet to transmit. A protocol in which each station knows the state of the message queues in all the other stations can be expected to be able to provide better performance.

The notion of guarantee is a characterizing feature of real-time virtual circuits and hence has to be accepted. However, the second reason, viz., the local view of the protocols, appears to be merely a restriction that arises because of the nature of the protocols that we have been using. Is it possible to eliminate this cause for pessimism through the use of a suitable reservation protocol? We examine various reservation protocols that have been proposed in the literature to try to answer this question.

7.2 Reservation Protocols

Various reservation protocols have been proposed in the context of broadcast satellite-based data communication and integrated voice data local networks. These come in different flavors. We examine below three classes of reservation protocols that have been proposed in the literature below.

7.2.1 Single Packet Reservation

In one scheme proposed by Kleinrock and Scholl [44], each packet transmission is preceded by a set of reservation slots (Figure 7.1). The length of each reservation slot is equal to the end-to-end propagation delay, and there is one reservation slot for every station in the system. If a station has a packet to transmit, it makes a reservation by broadcasting a burst of noise during its reservation slot. This burst is a signal to all the stations with a higher address to refrain from transmitting in the packet transmission slot following the reservation slots. Thus this protocol selects the station with the lowest address that has a packet for transmission. Figure 7.1

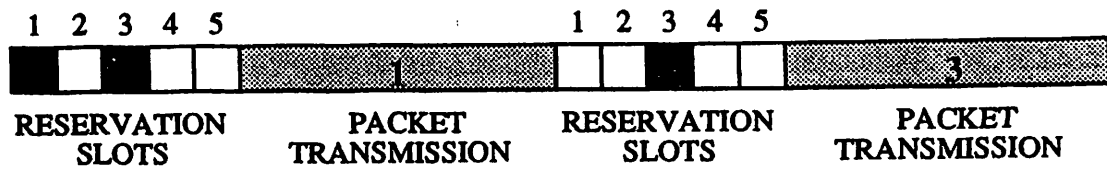


Figure 7.1 Bit-mapped reservation protocol

depicts a system with 5 nodes. In the first transmission cycle shown, there are two nodes (1 and 3) that have a packet to transmit. Node 1 being the station with the smaller address gets to transmit its packet. In the following transmission cycle, node 3 is the only node with a pending packet and is granted transmission rights by the protocol. This protocol makes a very limited amount of local information, viz., the information as to which nodes have a packet to transmit, global. However this information is not sufficient for a node to determine whether a packet that has just arrived will be able to meet its deadline or not. Hence this protocol is *not suitable* for providing real-time connection-oriented services.

7.2.2 Multiple Packet Reservations

The protocols that we look at in this section make reservations for multiple packets, thereby avoiding contention for the packets that follow the first one in a sequence of packets.

Binder [15] has proposed a protocol that normally operates like a TDMA protocol, where each slot is associated with a particular station (the "owner" of the slot). However, when a station has nothing to transmit, its slot is made available to other stations. Any station that requires the slot may capture it through a process of contention. This station then holds on to the slot as long as it has a packet to transmit (i.e., it has reserved this slot for its own use in the subsequent frames). If the owner of the captured slot needs the slot back in the mean time, it induces a collision by transmitting during the slot. The collision is a signal to the capturer that the owner needs the slot back. The capturer relinquishes the slot immediately.

	SLOT 1	SLOT 2	SLOT 3	SLOT 4	SLOT 5	SLOT 6
FRAME 1	1	2	3	collision	5	6
FRAME 2	idle	2	3	4	5	6
FRAME 3	collision	idle	3	4	5	6
FRAME 4	6	collision	3	idle	idle	6
FRAME 5	idle	collision	3	3	3	idle
FRAME 6	3	3	3	3	3	idle


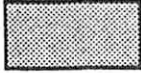
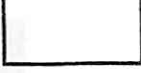
		
collision	transmission	idle

Figure 7.2 Binder's broadcast satellite protocol

Figure 7.2 depicts a system with 6 nodes and six corresponding slots. Node 1 has only one packet to transmit, which it transmits during its slot in frame 1. The fact that slot 1 is idle in the second frame is an indication to the nodes that the owner of this slot (node 1) has no more packets to transmit. So in frame 3, nodes 3 and 6 contend for this slot resulting in a collision. Due to the randomized back off algorithm, node 6 successfully captures the slot in frame 4. Node 6 has no packets to transmit in frame 5. So slot 1 again becomes available for contention in frame 6, when node 3 captures it. Note that, since node 3 always has a packet to transmit, slot 3 never comes up for contention. Slot 4 was presumably used by some other node prior to frame 1. In frame 1, node 4 recaptures its slot by causing a collision. Finally, slot 6 is empty in frame 6 because no nodes have a packet to transmit.

While Binder's protocol requires that the number of stations be known in advance, a related protocol, reservation ALOHA, proposed by Crowther et al. ([12],[27]) does not have such a requirement. In this protocol, slots do not have owners associated with them. Instead, whenever a station needs a slot, it waits for a slot to become idle, i.e., it waits for some station that is currently transmitting packets to complete its transmission. It then tries to capture the slot released by this station through a process of contention. The station that succeeds in capturing

	SLOT 1	SLOT 2	SLOT 3	SLOT 4	SLOT 5	SLOT 6
FRAME 1	15	7	6		10	2
FRAME 2		7	6	5	10	2
FRAME 3	12		6	5	10	2
FRAME 4	12		6	5		
FRAME 5	12		6	5	3	6
FRAME 6	12	13	6	5	3	

collision

transmission

idle

Figure 7.3 Reservation ALOHA

the slot keeps the slot until it has no more packets to transmit, at which point the slot becomes available again to anyone who needs it. This protocol has the potential problem that if a significant traffic imbalance exists even for a short time, it is possible for one station to capture the entire channel starving the other stations. Figure 7.3 illustrates the reservation ALOHA protocol. The system depicted in the figure has 6 slots and 15 nodes. Station 15 has completed its transmission in frame 1. The fact that slot 1 is idle in frame 2 indicates to all the stations that the previous holder of this slot has completed its transmission. This slot comes up for contention in frame 3, when station 12 captures it successfully. Slot 2, held by station 7 till frame 2, comes up for contention in frame 4. After two collisions, the slot is captured by station 13 in frame 6. Slot 3 is never relinquished by station 6, since it always has a packet to transmit. The events in the other slots and frames may be explained similarly.

Another protocol that is similar in spirit to the above protocols is the protocol proposed by Limb and Flamm for the Fasnet [60] local area network. Fasnet is a high bandwidth unidirectional dual bus system capable of supporting both voice and data traffic. In the protocol proposed in [60], when a user makes a phone call, the station tries to capture an empty voice slot and holds onto the slot for the duration of a

talk spurt (A conversation can be modeled as a series of talk spurts with intervening silence spurts). This slot, which constitutes a virtual voice channel, is relinquished during a silence spurt at which point it is made available to a data station. At the next talk spurt, the voice station captures another slot and this continues till the end of the call. The capture and release of slots is mediated through the two end stations on the bus. This protocol guarantees that once a call has been set up, the source station will always be able to find a free slot in each cycle of slots, whenever it needs one.

Note that, in fact, all the reservation protocols that we have described in this section, assume a traffic model similar to that of the Fasnets protocol, namely the model of a phone call. They assume that the traffic that originates at a node consists of transmission spurts followed by idle spurts (the Fasnets protocol, in addition, takes into account the real-time constraints of voice data). During a transmission spurt, a station has a sequence of messages or packets to transmit. Thus once a slot has been allocated to a station, it makes little sense to throw the slot open for contention until the station has transmitted all its packets (in an application like voice communication, owing to the real-time requirements of the data, such an action may not even be feasible). Therefore these protocols allocate a slot to a station for one transmission spurt, which may be of any length. The protocols do not involve the broadcast of any local information. Hence each station has to make worst case assumptions based only on local knowledge, if it wishes to guarantee asynchronous time-constrained messages.

7.2.3 Explicit Reservation Protocols

We next examine some protocols that make use of explicit reservation messages to arbitrate access to the channel. These protocols were proposed in the context of satellite-based packet-switched communication. All of them try to improve channel utilization and reduce the average delay experienced by a packet. The approach used basically consists of broadcasting short control messages requesting channel time prior to transmitting the actual queued messages.

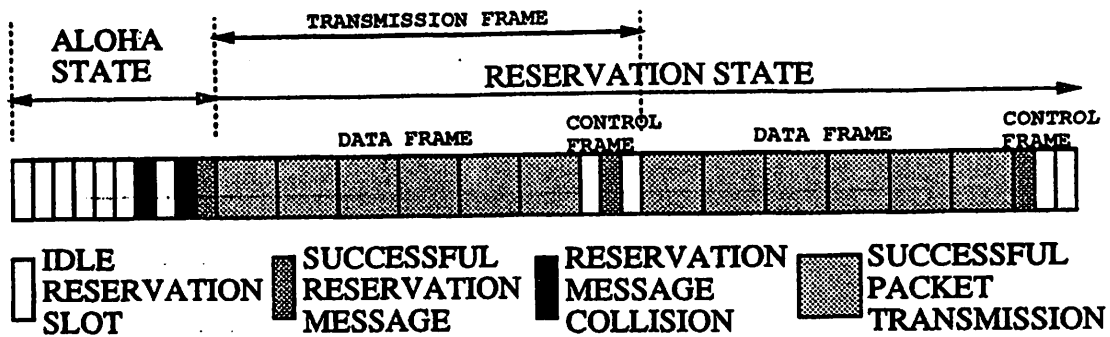


Figure 7.4 Roberts' reservation protocol

The first protocol in this category that we examine was proposed by Roberts ([77],[49]). In this protocol (Figure 7.4), the channel may be in one of two states (i) the *ALOHA state* or (ii) the *reservation state*. The channel is in the ALOHA state when there are no pending reservations waiting to be serviced. In this state, stations may transmit new *reservation* messages according to the slotted ALOHA protocol [78]. When a reservation message gets transmitted successfully, the channel shifts to the reservation state in which reservations get serviced. In the reservation state, each transmission frame (or cycle) consists of a data frame and a control frame. The data frame is divided into data slots each equal to one data packet transmission time in length. The control frame is divided into reservation slots each equal to one reservation message transmission time in length. The data slots are used by stations that have made reservations, while the reservation slots are used by stations to make new reservations. Each reservation message consists of a number between 1 and 8 that indicates how many packets the station has to transmit. Since the channel is a broadcast channel, every reservation message is received by all stations. Each station keeps track of the total number of packets for which reservations have been made and the position of its own reservation message (according to the FIFO ordering) relative to the reservation currently being serviced. When a station's reservation

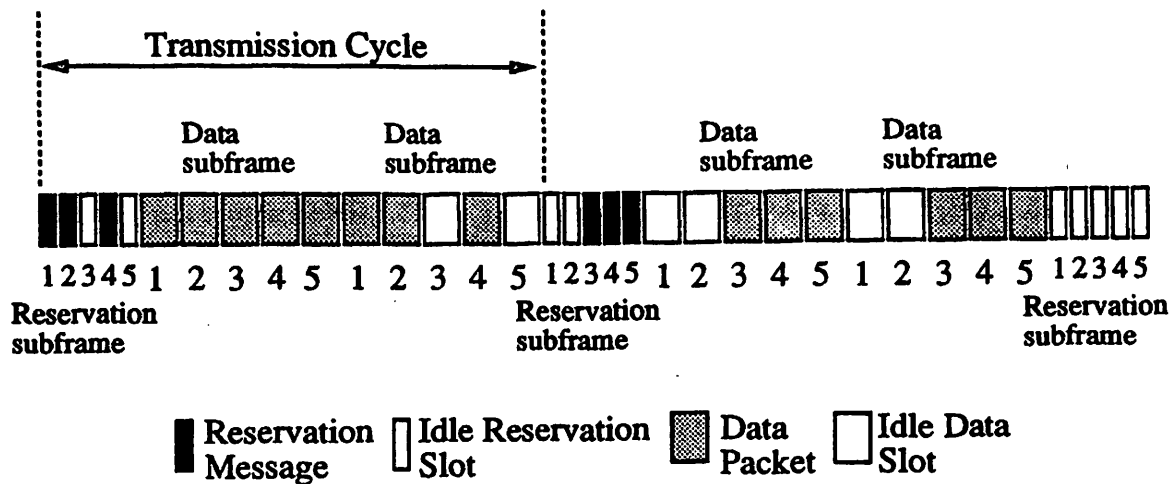


Figure 7.5 Reservation TDMA

request is ahead of all the pending reservation messages, the station transmits the corresponding data packet (for which the reservation was made).

The reservation TDMA protocol (Figure 7.5), proposed by Weissler et al. ([103],[49]), is a modification of the protocol proposed by Binder that we described in Section 7.2.2. In the modified protocol, each transmission cycle consists of k data subframes preceded by a reservation subframe, where k is a system parameter ($k=2$ in Figure 7.5). Each of these subframes consists of one slot per station (of which the corresponding station is the "owner"). The length of a slot in the reservation subframe is equal to the transmission time of a reservation packet, while the length of a message subframe slot is equal to that of a data packet. During its reservation slot, a station *broadcasts* a reservation message that contains a "new message count", i.e., the number of new packets (which may be zero) that have arrived since the last time it made a reservation. At a commonly agreed upon time known as the update slot, each node updates its local copy of a *reservation table* based on the received new message counts. This table is used to assign data slots to the nodes according to the following algorithm (which every station executes). If the owner of a slot has pending reservations in the table, then the slot is assigned to the owner; otherwise, the slot is assigned to other stations that have made reservations in a

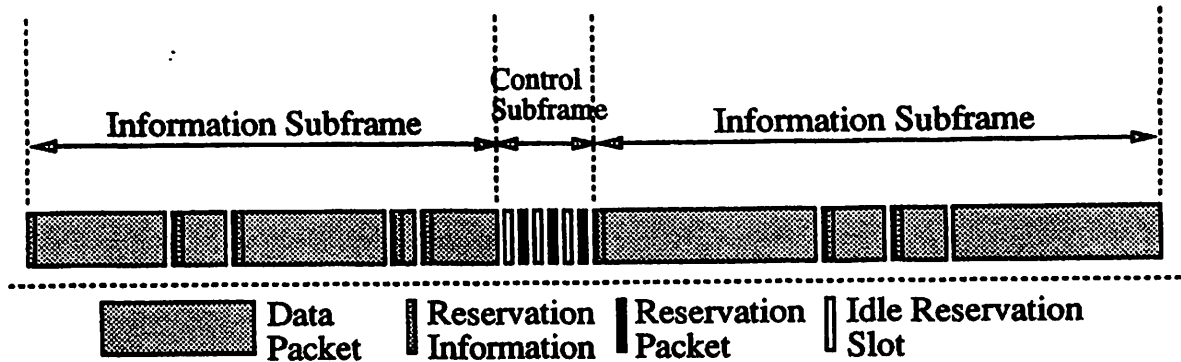


Figure 7.6 Priority-oriented demand assignment

round-robin manner. In Figure 7.5, stations 1, 2 and 4 transmit reservation request during their respective reservation subframe slots in the first transmission cycle¹. Station 1 makes a request for 4 data slots, while stations 2 and 4 make requests for 2 data slots each. Since stations 3 and 5 do not have any packets to transmit, the slots owned by them in the first data subframe are allocated to station 1 to satisfy its requirements. In the second transmission cycle, stations 3, 4 and 5 each make reservation requests for two packets and are allocated the slots owned by them.

The last explicit reservation protocol that we look at was proposed by Jacobs et al. ([39], [49]). This protocol known as PODA (priority-oriented demand assignment) is a general-purpose protocol that is capable of supporting both synchronous (stream) and asynchronous (datagram) traffic, multiple delay classes and priorities, and variable message lengths. In this protocol, channel time is divided into frames

¹This figure is merely meant to be illustrative. In the actual protocol, due to the large end to end propagation delay (≈ 250 millise) in satellite based communication, reservations for one transmission cycle are made in the reservation subframe of the preceding transmission cycle

each of which consists of an *information subframe* and a *control subframe* (Figure 7.6). The information subframe is used for scheduled datagram and stream² transmissions. These transmissions may contain additional reservation requests in their headers. The control subframe is used to broadcast reservations that cannot be sent as part of the data transmissions in a timely manner (e.g., when a station wants to make a reservation for a high priority message or when a station wants to join the system). The PODA protocol comes in two flavors, each characterized by a different access method during the reservation subframe. The CPODA (contention PODA) protocol makes use of a random access scheme similar to that of Robert's protocol to arbitrate access to the channel for reservation messages. The FPODA (fixed assignment PODA) uses a TDMA scheme similar to that in the protocol proposed by Weissler et al. for this purpose.

In the PODA protocol, all stations³ maintain a local copy of a scheduling queue ordered by reservation urgency, which is a function of the delay class and priority of a message. Reservations with the same urgency are ordered further to ensure fairness to all the stations involved. Channel time in the information subframe is allocated to the stations according to this queue. Whenever a reservation request for an asynchronous message is broadcast, all the nodes that successfully receive the message enter the request in their respective local scheduling queues (on the basis of its reservation urgency). A reservation for a stream is made when the stream is set up and is entered into a separate stream queue that is maintained at each station. A stream reservation message contains information regarding the stream repetition interval, desired maximum delay relative to this interval, and priority. Whenever the repetition time is near, each station creates a new reservation request for the stream's next message and enters it into its local scheduling queue. The PODA scheme is the most general scheduling scheme (based on individual message requirements) among the three explicit reservation schemes that we have looked

²A stream is a virtual channel, e.g., a voice channel, with a constant message arrival rate. Only one reservation message is required for all the messages that are part of the stream and is broadcast when the stream is set up.

³The protocol also permits only a subset of stations to perform the distributed scheduling - the other stations depend on centralized assignment in this case.

at and does *scheduling based on a global view of all messages in the system*. This property is exactly what we would like to have in a protocol used to implement real-time connection-oriented service.

Note that, all the reservation protocols that we have examined so far are essentially non-real-time protocols. These protocols evolved from a desire to overcome the disadvantages of TDMA at low loads and ALOHA at high loads. Their main goal is to improve *channel utilization* (the throughput achievable with the slotted ALOHA protocol is limited to a maximum of 36 %) and reduce *average delay*. They are based on the well known performance improvement (in terms of average delay) resulting from combining multiple single server queues into a single server queue with the combined service rates of all the queues. However it should be kept in mind that we are interested in *real-time* protocols; the performance measure that is of interest to us is the *guarantee ratio* for asynchronous real-time virtual circuit messages.

7.3 Guarantee-Based Reservation Protocols

The individual-message-based global scheduling approach used in the PODA protocol appears to be a good basis for developing a guarantee-based reservation scheme. For example, each station can transmit reservation messages periodically and as part of data messages. These reservation messages will contain the arrival time and deadline requirements of packets that have arrived since the last reservation. Each station then applies the same guarantee algorithm on these reservation requests. If it is possible to guarantee the deadline of a packet, then a reservation is made for the packet in the local copy of the channel schedule at each station. Stations then transmit packets on the basis of their local copies of the channel schedule.

Unfortunately such a guarantee-based protocol based on a common channel schedule is not realizable in practice for the following reason. In all explicit reservation schemes, reservation messages may be corrupted by noise on the channel. This may result in different stations receiving different messages. As a result, the

local scheduling queue or reservation table at each station may become mutually inconsistent. Since each station decides when to transmit on the channel based on this table, this loss of scheduling synchronization could potentially lead to collisions (when two different stations believe it is their turn to transmit) or unutilized reserved time (when each station believes it is some other station's turn to transmit). This problem is typically solved by one of the following schemes:

1. Each data message carries control information about the source station's current view of the scheduling information. For example, in Robert's scheme, information about the number of reservations that have not yet been serviced is carried in each data packet. If a station detects an inconsistency between this information and its own local view, then it remakes reservations.
2. Each station monitors the channel and compares the actual transmissions to its own view of the channel schedule, as is done in PODA. If it detects inconsistencies, it switches to a state where it suspends its transmissions and tries to reestablish a local view that is consistent with the local view of the schedule at the other nodes. In order to reestablish consistency, it may have to cancel its existing reservations and remake reservations.

Cancellation and remaking of reservations will result in increased delay. Thus the presence of channel noise has the effect of merely increasing the average delay of packets in these schemes.

For a communication scheme based on the notion of *guarantee*, once a station loses scheduling synchronization, it may transmit during the turn of another station resulting in a collision and thus cause a *violation of the guarantee* (an event that contradicts the notion of guarantee) that was provided to the colliding packets. The guarantees provided to the packets that were scheduled to be transmitted during the period when the station tries to reacquire synchronization are also violated, since the station cannot transmit any packets during this period. Such potential guarantee violations cannot be compensated for by guaranteeing multiple (redundant) packets, since once scheduling synchronization is lost the guarantees for the redundant copies may also be violated. Thus reservation-based guarantee schemes are

difficult to realize in practice, since they require the *continuous* maintenance of the consistency of the local copies of a common data structure (the channel schedule); this requires that *all* the stations always receive *every* update (reservation) message *without errors*, which it may not be possible to guarantee in practice without the use of special schemes for fault-tolerance.

One approach to improving the system's tolerance of errors and failures is the standard approach based on redundancy. The shared bus is replicated n times with the same MAC protocol being employed on all the buses. There are n copies of the global queue at each node, one associated with each bus. The queue associated with a bus is constructed entirely using the reservation requests that are received on the bus. This ensures that as long as there is at least one copy of the queue that is consistent across all the nodes, the system will function correctly.

An alternative approach is to employ a synchronous atomic broadcast protocol [21], again based on redundancy. Such a protocol guarantees the following three properties:

1. *Atomicity*: Every message whose broadcast is initiated by a sender is either delivered to all receivers or to none.
2. *Order*: All delivered messages are received in the same order at all receiving nodes.
3. *Termination*: Every message broadcast by a sender and delivered to some correct receiver, is delivered to all correct receivers after some known time interval.

These properties and the fact that the scheduling algorithms executed at each node are identical ensure that the local copies of the global queue are mutually consistent in spite of errors.

Owing to the complexities and difficulties in realizing the reservation approach, we did not consider the reservation approach further in this research.

7.4 Summary

We identified an important shortcoming of various protocols that we previously considered, viz., their pessimism that arises because of their support for the notion of guarantee and their localized knowledge. We examined three classes of reservation-based protocols to identify protocols that operate on the basis of global knowledge. The explicit reservation protocols, especially the PODA protocol, possess this property, and hence could be potential candidates for implementing a guarantee-based protocol that operates on the basis of global knowledge. However there is one problem that is common to all explicit reservation based protocols, viz., the possibility of loss of scheduling synchronization due to channel noise. In the case of non real-time protocols in which the quality of service is measured by the average delay, loss of scheduling synchronization may result in increased delays. However, in the case of real-time protocols that support the notion of guarantee, loss of scheduling synchronization is not permissible, since this may result in the violation of existing guarantees. Therefore, explicit reservation-based schemes that make use of global knowledge are not suitable for implementing guarantee-based services, unless additional schemes for fault-tolerance are employed.

CHAPTER 8

CONCLUSION

In this chapter, we summarize the contributions of this research and identify directions for future work.

8.1 Results and Contributions

This research attacked two important problems that arise in distributed real-time systems and has generated novel solutions to the problems. The contributions of this research include a new protocol for clock synchronization based on a probabilistic approach, development of several new real-time communication services (RTCOS and RTCLS) and abstractions (real-time virtual circuit and real-time datagram), a new local area network architecture (RTLAN) for distributed real-time systems and a number of new real-time medium access control protocols (PRI, RTDG, RTVC, INTPVC and INTPDG) for supporting the new classes of services and abstractions. An integral part of the research is the validation of the proposed protocols through analysis and extensive simulation studies. Below, we briefly summarize the contributions and results of this research.

8.1.1 Clock Synchronization

We developed *a new protocol for clock synchronization* in distributed real-time systems that can achieve smaller clock skews than most existing protocols. Due to the constraints imposed by Eq. (1.2) on the maximum clock skew that can be guaranteed with certainty, we took a probabilistic approach to the problem of clock synchronization. The clock synchronization algorithm proposed was shown to perform much better than the deterministic algorithms that are constrained to obey

Eq. (1.2). We demonstrated, in our examples, that our algorithm is capable of guaranteeing a maximum deviation between clocks of the order of a few milliseconds, compared to the smallest maximum deviation of 50 milliseconds that can be guaranteed by deterministic algorithms. However the guarantee offered by our algorithm is probabilistic, i.e., there is a non-zero probability that the guarantee offered by our algorithm will fail to hold. The probability of invalidity however can be made *extremely* small by transmitting a sufficient number of messages. In our examples, we were able to achieve a probability of invalidity of (1×10^{-9}) by transmitting 10 synchronization messages. We also showed that the performance of our algorithm is comparable to the performance of the probabilistic algorithm proposed by Cristian [22]. In addition, unlike Cristian's algorithm in which each slave is individually responsible for synchronizing with the master node, in our algorithm (in which the master is responsible for getting the slaves to synchronize with itself) the number of messages required to achieve a required maximum clock skew can be further reduced by a factor equal to the number of nodes in the system, if a broadcast channel is used.

We also presented a detailed analysis of the proposed clock synchronization protocol. Among other things, we derived an expression for the minimum number of synchronization messages required to guarantee a specified maximum deviation at resynchronization and a specified probability of invalidity. We considered three analytical bounds on the probability of invalidity, viz., the *Tchebysheff*, *error function* and *exponential* bounds, and showed that the probability of invalidity decreases exponentially (or better) with the number of synchronization messages.

8.1.2 RTLAN

Most current work in real-time communication deals with static systems in which messages with timing constraints are mainly periodic, or occur only rarely. However the dynamic closed loop distributed real-time systems of the future will be characterized by richer communication patterns. Specialized network services and

protocols will be required to support the communication requirements of these systems. In this dissertation, we proposed RTLAN, *a new local area network architecture* for communication in such systems. RTLAN is targeted for complex real-time applications which have time-constrained communication requirements that range from simple best effort delivery requirements to dynamic guarantees of general timing requirements. Some of the distinguishing features of RTLAN include the provision of *new classes of connection-oriented and connectionless services* known as real-time connection-oriented service (RTCOS) and real-time connectionless service (RTCLS) respectively, that take the timing constraints of messages explicitly into account, incorporation of the notions of *a priori* scheduling and guarantee in the logical link control layer, and the use of specialized real-time protocols at the medium access control layer. We developed the useful *abstractions* of real-time virtual circuits and real-time datagrams in the context of a local area network.

8.1.3 Real-Time MAC Protocols

We considered a distributed real-time system based on a local area network with a *bus* topology, and proposed a *new homogeneous suite of real-time medium access control protocols* for supporting RTCOS and RTCLS, that is devoid of deficiencies found in existing standards for medium access control protocols for bus-based systems, viz., the IEEE 802.3 standard and the IEEE 802.4 standard. The proposed suite consists of five protocols, viz., PRI, RTDG, RTVC, INTPVC and INTPDG, all based on a uniform window splitting paradigm. The protocol PRI implements priority-based arbitration. The protocol RTDG implements real-time datagram arbitration based on the minimum laxity first policy. The protocol RTVC implements real-time virtual circuit arbitration. The protocols INTPVC and INTPDG are integrated protocols that may be used to support both real-time datagram and real-time virtual circuit arbitration in an integrated manner on a common bus. We analyzed the window protocols and derived various properties exhibited by the protocols. We also conducted a simulation study to evaluate the performance of the protocols. The

main properties of the protocols, derived from the analytical and simulation study are summarized below:

1. The per packet contention resolution overhead for the window-based contention resolution procedure used by the window protocols increases logarithmically with the size of the window.
2. The PRI window protocol not only provides a capability, viz., system-wide packet level priority-based arbitration, that the 802.3 and 802.4 standard protocols lack, but also is characterized by a slightly smaller average priority resolution overhead than other non-standard protocols that have been proposed for priority resolution.
3. The RTDG protocol closely approximates the minimum laxity first policy for channel arbitration.
4. The performance of the RTDG window protocol is superior to that of an idealized baseline protocol that provides a bound on the performance achievable by any non-real-time protocol (i.e., a protocol that has no notion of packet timing constraints).
5. The performance of the RTVC window protocol is close to that of an idealized baseline protocol that provides a bound on the performance achievable by any protocol that can guarantee bounded channel access times. The performance of the RTVC window protocol is also observed to be better than that of the TDMA protocol under certain conditions.
6. The integrated window protocol INTPVC is characterized by a smaller (by a factor of about 2) worst case channel access time for real-time virtual circuit packets, than the standard 802.4 token bus protocol operating in integrated mode (priority mode).
7. The quality of service provided to real-time virtual circuit packets by the integrated window protocol INTPVC is better than that provided by VCTIMER, an idealized baseline integrated protocol.

8. The integrated window protocol INTPDG is characterized by a worst case channel access time for real-time virtual circuit packets, that is about the same as (but it is larger by about one packet transmission time than) that for the standard 802.4 token bus protocol operating in integrated mode.
9. Use of an integrated protocol results in an improvement in the quality of performance. INTPVC provides a better quality of performance for real-time virtual circuit packets than RTVC. INTPDG provides a better quality of performance for real-time datagram packets than RTDG.

The protocol suite that we have proposed in this dissertation would be an excellent choice for a builder of a distributed real-time application with dynamic predictability requirements, based on a bus. In addition to overcoming deficiencies of existing standards for bus-based systems, and to the good performance characteristics pointed out above, the protocol suite proposed here also has the advantage of structural homogeneity (thereby enabling the use of a uniform medium access control logic and LAN controller hardware), since all the protocols are based on a uniform window splitting paradigm.

8.2 Future Work

A number of directions for future research, that have roots in the work presented in this dissertation, can be identified. We briefly discuss these potential directions below.

- **Clock Synchronization**

There are a number of aspects of our clock synchronization protocol that can be improved through further research. Unlike most other work in this area, whose main focus is on fault-tolerance, the focus of our work has been on achieving a smaller maximum deviation between clocks in a distributed system, than previously accepted bounds. Improvement of the fault-tolerance of our algorithm, by identifying new schemes to handle various kinds of failures,

is one area of research. Another area of research is to identify how the time transmission protocol proposed in this paper could be combined with the algorithms in [26], [31], [47], [48], [54], [53] and [63] to overcome the limitations imposed by Eq. (1.2). Finally, it would be interesting to identify other kinds of probabilistic algorithms, and to determine the theoretical and practical bounds on the maximum deviation between clocks in a distributed system, that can be guaranteed by probabilistic clock synchronization protocols.

- **Interprocess Communication**

We delimited the scope of our research to *protocols* for distributed real-time systems. Thus we did not consider application layer issues in this research, in any depth. An important application layer issue is support for predictable interprocess communication. The realization of predictable interprocess communication with RTLAN as the underlying network architecture is an area that offers scope for further research.

- **Alternative Topologies:**

Local area networks based on a bus topology are very common in today's distributed real-time systems and will continue to remain so for the foreseeable future. Point-to-point network topologies such as mesh connected networks that have good fault-tolerance properties are beginning to be investigated ([41]). Such alternative topologies offer scope for research in real-time communication.

- **Alternative Paradigms:**

In this work, we followed the most prevalent paradigm, viz., *non-shared memory*, that is to be found in distributed real-time systems. Most current work on real-time communication assumes this paradigm. Recently, there has been an interest in an orthogonal paradigm, viz., *shared memory*. The seemingly inconsistent notions of a globally *shared memory* and the *physical distribution* inherent in a distributed real-time system are realized in the same system through the use of *memory networks*.

In a memory network (also known as *replicated shared memory* or *reflective memory*), memory units are networked through a high speed interconnect. An update to a location in one memory unit is immediately and automatically propagated to the corresponding locations in the other units through the high speed interconnect and appropriate hardware. This ensures that corresponding memory locations on the different memory units are mutually consistent except for the small latency introduced by the interconnect, thus creating the illusion of a globally shared memory. An example of a memory network is the SCRAMNet network [5] marketed by Systran Corporation. SCRAMNet uses a fiber optic register insertion ring to interconnect the various memory units. Memory networks give rise to a number of interesting avenues of exploration. Investigation of suitable topologies and protocols for the high speed interconnect is one such avenue. For example, how would a star topology with a high-speed switch compare with the register insertion ring proposed for SCRAMNet? The study of whether and how the communication services and abstractions developed in this dissertation may be adapted to suit memory networks is another potential area of research. Propagation of updates to a 32 bit memory location in SCRAMNet involves the transmission of 51 bits of overhead. An interesting question for investigation is whether it is possible to use some form of "caching" to reduce this overhead.

- **Multimedia Communication:**

In this research, we restricted the term *real-time communication* to mean communication in a distributed real-time system and proposed abstractions, services and protocols in the context of a local area network. The term *real-time communication* is more broadly used to denote any communication activity that is characterized by timing constraints. For example, the transmission of packetized voice and video data is characterized by constraints on the maximum delay of the packets and variance of this delay. Integrated voice/data communication ([30], [61],[68]) and kernel and communication support for WAN-based distributed multimedia systems ([3], [4]) are areas that we would like to investigate further.

BIBLIOGRAPHY

- [1] Abramson, N. The Aloha System — Another Alternative for Computer Communications. *Proceedings of the AFIPS Fall Joint Computer Conference*, 37, Fall 1970.
- [2] Alger, L.S., Lala, J.H. A Real-Time Operating System for a Nuclear Power Plant Computer. *Proc. Real-Time Systems Symposium*, December 1986.
- [3] Anderson, D.P., Tzou, S., Wahbe, R., Govindan, R., Andrews, M. Support for Continuous Media in the DASH System. *Proc. 10th International Conference on Distributed Computing Systems*, Paris, France, May/June 1990.
- [4] Anderson, D.P., Herrtwich, R.G., Schaefer, C. SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet. Report No. UCB/CSD 90/562, February 1990.
- [5] Arment, J.F., Marquart, J., Trainor, W.L. Replicated Shared Memory for Multi-Vendor Networks. *Defense Computing*, February 1990.
- [6] Arvind, K. A New Probabilistic Algorithm for Clock Synchronization. *COINS Technical Report TR 89-86*. Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, January 1990. Submitted to the *IEEE Transactions on Parallel and Distributed Systems*.
- [7] Arvind, K. A New Probabilistic Algorithm for Clock Synchronization. *Proc. of the Real-Time Systems Symposium*, Santa Monica, CA, December 1989.
- [8] Arvind, K. Distributed Robot Control. Unpublished Manuscript, December 1988.
- [9] Arvind, K., Ramamritham, K., Stankovic, J.A. A Local Area Network Architecture for Communication in Distributed Real-Time Systems. To Appear in *Real-Time Systems*, Vol. 3, No. 2, May 1991.
- [10] Arvind, K., Ramamritham, K., Stankovic, J.A. Window MAC Protocols for Real-Time Communication Services. *Submitted for Publication*, January 1991. Also Available as COINS TR 90-127.
- [11] Bares, J., Hebert, M., Kanade, T., Krotkov, E., Mitchell, T., Simmons, R., Whittaker, W. Ambler, An Autonomous Rover for Planetary Exploration. *Computer*, Vol. 22., No. 6, June 1989, pp. 18-26.

- [12] Bertsekas, D., Gallager, R. *Data Networks*. Englewood Cliffs, N.J.: Prentice Hall Inc., 1987.
- [13] Bihari, T.E., Walliser, T.M., Patterson, M.R. Controlling the Adaptive Suspension Vehicle. *Computer*, Vol. 22., No. 6, June 1989, pp. 59-65.
- [14] Binder, R. Packet Protocols for Broadcast Satellites. In *Protocols and Techniques for Data Communication Networks*, ed. F.F. Kuo. Englewood Cliffs, N.J.: Prentice Hall Inc., 1981.
- [15] Binder, R. A Dynamic Packet Switching System for Satellite Broadcast Channels. *Proc. ICC*, 1975, pp. 41-1 to 41-5a.
- [16] Capetanakis, J.I. Generalized TDMA: The Multi-Accessing Tree Protocol. *IEEE Transactions on Communications*, Vol. COM-27, No. 10, October 1979.
- [17] Carlow, G.D. Architecture of the Space Shuttle Primary Avionics Software System. *Communications of the ACM*, 27(9):926-36, September 1984.
- [18] Cheng, S., Stankovic, J.A., Ramamritham, K. Scheduling Algorithms for Hard Real-Time Systems - A Brief Survey. *Tutorial Hard Real-Time Systems*, IEEE Computer Society Press, 1988, pp. 150-173.
- [19] Chlamtac, I., Franta, W.R., Levin, D. BRAM: The Broadcast Recognizing Access Method. *IEEE Transactions on Communications*, Vol. COM-27, No. 8, August 1979.
- [20] Comer, D.E., Yavatkar, R. FLOWS: Performance Guarantees in Best Effort Delivery Systems. *Proc. IEEE INFOCOM*, 1989.
- [21] Cristian, F. Synchronous Atomic Broadcast for Redundant Broadcast Channels. *Real-Time Systems*, Vol. 2, No. 3, September 1990, pp. 195-212.
- [22] Cristian, F. Questions to Ask When Designing or Attempting to Understand a Fault-Tolerant Distributed System. Keynote Address, *Third Brazilian Conference on Fault-Tolerant Computing*, Rio de Janeiro, September 1989.
- [23] Cristian, F. A Probabilistic Approach to Distributed Clock Synchronization. *Distributed Computing*, 3:146-158, 1989.
- [24] Cristian, F. *Private Communication*. December 1989.
- [25] Cristian, F., Dancey, R.D., Dehn, J. Fault-Tolerance in the Advanced Automation System. *IBM Research Report RJ 7424 (69595)*, April 1990.
- [26] Cristian, F., Aghili, H., Strong, R. Clock Synchronization in the Presence of Omission and Performance Faults, and Processor Joins. *Proc. of the International Conference on Fault-Tolerant Computing*, 1987.

- [27] Crowther, W., Rettburg, R., Walden, D., Ornstein, S., Heart, F. A System for Broadcast Communication: Reservation Aloha. *Proc. 6th Hawaii International Conference on System Sciences*, 1973, pp. 371-374.
- [28] Dolev, D., Halpern, J.Y., Strong, H.R. On the Possibility and Impossibility of Achieving Clock Synchronization. *Journal of Computer and System Sciences*, Vol. 32, No. 2, 1986, pp. 230-250.
- [29] Franta, W.R., Bilodeau, M.B. Analysis of a Prioritized CSMA Protocol Based on Staggered Delays. *Acta Informatica*, Vol. 13, 1980, pp. 299-324.
- [30] Gonsalves, T. Packet-Voice Communication on an Ethernet Local Computer Network: An Experimental Study. *Proc. of the ACM SIGCOMM Communications Architectures and Protocols Symposium*, Austin, TX, March 1983.
- [31] Halpern, J., Simons, B., Strong, R. Fault-Tolerant Clock Synchronization. *Proc. of the Third Annual ACM Symposium on Principles of Distributed Computing*, August 1984.
- [32] Hutchison, D. *Local Area Network Architectures*. Addison-Wesley, 1988.
- [33] *IEEE Standards for Local Area Networks. Token-Passing Bus Access Method*. New York: IEEE, 1982.
- [34] *IEEE Standards for Local Area Networks: Logical Link Control*. New York: IEEE, 1984.
- [35] *IEEE Standards for Local Area Networks: Carrier Sense Multiple Access with Collision Detection*. New York: IEEE, 1985.
- [36] *IEEE Standards for Local Area Networks. Token Ring Access Method*, New York: IEEE, 1986.
- [37] Iida, I., Ishizuka, M., Yasuda, Y., and Onoe, M. Random Access Packet Switched Local Computer Network with Priority Function. *Proceedings National Telecommunications Conference*, December 1980, pp. 37.4.1-37.4.6.
- [38] Iyengar, S.S., Kashyap, R.L. Autonomous Intelligent Machines, *Computer*, Vol. 22, No. 6, June 1989.
- [39] Jacobs, I.M., Binder, R., Hoversten, E.V. General Purpose Packet Satellite Networks, *Proc. of the IEEE*, 1978, pp. 1448-1468.
- [40] Kallmes, M.H., Towsley, D., Cassandras, C.G. Optimality of the Last-In-First-Out (LIFO) Service Discipline in Queueing Systems with Real-Time Constraints. *Proceedings of the 28th Conference on Decision and Control (CDC)*, Tampa, Florida, 1989, pp. 1073-1074.

- [41] Kandlur, D.D., Kiskis, D.L., Shin, K.G. HARTOS: A Distributed Real-Time Operating System. *Operating Systems Review*, August 1989.
- [42] Kleinrock, L. *Queueing Systems*. Vol. I, 1976.
- [43] Kleinrock, L., Tobagi, F. Packet Switching in Radio Channels: Part I-Carrier Sense Multiple Access Modes and their Throughput-Delay Characteristics. *IEEE Transactions on Communications*, COM-23, 12, December 1975, pp. 1015-1029.
- [44] Kleinrock, L., Scholl, M.O. Packet Switching in Radio Channels: New Conflict-Free Multiple Access Schemes. *IEEE Transactions on Communications*, COM-28, 7, July 1980, pp. 1015-1029.
- [45] Knightson, K.G., Knowles, T., Larmouth, J. *Standards for Open Systems Interconnection*. New York: McGraw-Hill Book Company, 1988.
- [46] Kopetz, H. Real Time in Distributed Real Time Systems. *Proc. IFAC Distributed Computer Control Systems*, 1983.
- [47] Kopetz, H. and Ochsenreiter, W. Clock Synchronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, 36(8):933-40, August 1987.
- [48] Kopetz, H., Damm, A., Koza, Ch., Mulazzani, M., Schwabl, W., Senft, Ch., Zainlinger, R. Distributed Fault-Tolerant Real-Time Systems: The MARS Approach. *MARS Report Nr. 4/88*, Institut für Technische Informatik, Technische Universität Wien, Austria, 1988.
- [49] Kuo, F.F. *Protocols and Techniques for Data Communication Networks*, Englewood Cliffs, N.J.: Prentice-Hall, 1981.
- [50] Kurose, J.F., Schwartz, M., Yemini, Y. Multiple-Access Protocols and Time-Constrained Communication. *Computing Surveys* 16(1):43-70, March 1984.
- [51] Kurose, J.F., Schwartz, M., Yemini, Y. Controlling Window Protocols for Time-Constrained Communication in a Multiple Access Environment. *Proc. Eighth IEEE International Data Communications Symposium*, October 1983.
- [52] Lamport, L. Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM*, Vol. 21, No. 7, July 1978, pp. 558-565.
- [53] Lamport, L., Melliar-Smith, P.M. Synchronizing Clocks in the Presence of Faults. *Journal of the Association for Computing Machinery*, Vol. 32, No. 1, January 1985, pp. 52-78.
- [54] Lamport, L., Melliar-Smith, P.M. Byzantine Clock Synchronization. *Proc. of the Third Annual ACM Symposium on Principles of Distributed Computing*, August 1984.

- [55] Lark, J.S., Erman, L.D., Forrest, S., Gostelow, K.P., Hayes-Roth, F., Smith, D.M. Concepts, Methods, and Languages for Building Timely Intelligent Systems. *The Journal of Real-Time Systems*, Vol. 2, 1990, pp. 127-148.
- [56] LeBlanc, T.J., Markatos, E.P. Operating System Support for Adaptable Real-Time Systems. *Proc. Seventh IEEE Workshop on Real-Time Operating Systems and Software*, May 1990.
- [57] Lehoczky, J.P., Sha, L. Performance of Real-Time Bus Scheduling Algorithms. *ACM Performance Evaluation Review*, Special Issue 14(1), May 1986.
- [58] Le Lann, G. The 802.3D Protocol: A Variation on the IEEE 802.3 Standard for Real-Time LANs. INRIA-BP 105, F-78153 Le Chesnay Cedex, France, July 1987.
- [59] Le Lann, G. Real-Time Protocols. *Local Area Networks, An Advanced Course*. Hutchison, D., et al, Editors, Springer Verlag, 1983.
- [60] Limb, J.O, Flamm, L.E. A Distributed Local Area Network Packet Protocol for Combined Voice and Data Transmission. *Advances in Local Area Networks*, IEEE Press, 1987.
- [61] Limb, J.O, Flores, C. Description of Fasnet — A Unidirectional Local Area Communication Network. *The Bell System Technical Journal*, Vol. 61, No. 7, September 1982.
- [62] Liu, C.L, Layland, J.W. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the Association for Computing Machinery*, Vol. 20, No. 1, 1973, pp. 46-61.
- [63] Lundelius, J., Lynch, N. A New Fault-Tolerant Algorithm for Clock Synchronization. *Proc. of the Third Annual ACM Symposium on Principles of Distributed Computing*, August 1984.
- [64] Lundelius, L., Lynch, N. An Upper and Lower Bound for Clock Synchronization. *Information and Control*, Vol. 62, 1984, pp. 190-204.
- [65] Malcolm, N., Zhao, W., Barter, C. Guarantee Protocols for Communication in Distributed Hard Real-Time Systems. *Proc. IEEE INFOCOM*, 1990.
- [66] Marinescu, D.C., Szpankovski, W. A Safe-State Approach to Real-Time System Scheduling. *Sixth IEEE Workshop on Real-Time Operating Systems and Software*, Pittsburgh, May 1989.
- [67] Martin, J. *Design of Real-Time Computer Systems*. Englewood Cliffs, N.J.: Prentice-Hall, 1967.

- [68] Maxemchuk, N.F. A Variation on CSMA/CD that Yields Movable TDM Slots in Integrated Voice/Data Local Networks. *The Bell System Technical Journal*, Vol. 61, No. 7, September 1982, pp. 1527-1550.
- [69] Metcalfe, R.M., Boggs, D.R. Ethernet: Distributed Packet Switching for Local Computer Networks. *Communications of the ACM*, July 1976, Vol. 19, No. 7, pp. 395-404.
- [70] Molle, M.L., Kleinrock, L. Virtual Time CSMA: Why Two Clocks are Better than One. *IEEE Transactions on Communications*, Vol. COM-33, No. 9, September 1985.
- [71] Muratore, J.F., Heindel, T.A., Murphy, T.B., Rasmussen, A.N., McFarland, R.Z. Real-Time Data Acquisition at Mission Control. *Communications of the ACM*, Vol. 33, No. 12, December 1990, pp. 18-31.
- [72] Nielsen, K. *Ada in Distributed Real-Time Systems*. New York: McGraw-Hill Book Company, 1990.
- [73] Panwar, S.S., Towsley, D., Wolf, J.K. Optimal Scheduling Policies for a Class of Queues with Customer Deadlines to the Beginning of Service. *Journal of the Association for Computing Machinery*, Vol. 35, No. 4, October 1988.
- [74] Ramamritham, K. Channel Characteristics in Local-Area Hard Real-Time Systems. *Computer Networks and ISDN Systems*, Vol 13, 1987, pp. 3-13.
- [75] Ramamritham, K., Stankovic, J.A., Zhao, W. Distributed Scheduling of Tasks with Deadlines and Resource Requirements. *IEEE Transactions on Computers*, Vol. 38, No. 8, August 1989, pp. 1110-1123.
- [76] Ramamritham, K., Stankovic, J.A. Time-Constrained Communication Protocols for Hard Real-Time Systems. *Sixth IEEE Workshop on Real-Time Operating Systems and Software*, Pittsburgh, PA, May 1989.
- [77] Roberts, L. Dynamic Allocation of Satellite Capacity through Packet Reservation. *Proc. NCC*, 1973, pp. 711-716.
- [78] Roberts, L. ALOHA Packet System with and without Slots and Capture. *ACM SIGCOMM Computer Communications Review*, Vol. 5, No. 2, April 1975.
- [79] Ross, F.E. FDDI - a Tutorial. *IEEE Communications Magazine*, Vol. 24, No. 5, May 1986.
- [80] Ross, F.E. An Overview of FDDI: The Fiber Distributed Data Interface. *IEEE Journal on Selected Areas in Communications*, Vol. 7, No. 7, September 1989.
- [81] Rothausser, E.H., Wild, D. MLMA-A Collision-Free Multi-Access Method. *Proc. IFIP Congress 77*, 1977.

- [82] Rouse, W.B., Geddes, N.D., Hammer, J.M. Computer-Aided Fighter Pilots. *IEEE Spectrum*, Vol. 27, No. 3, March 1990.
- [83] Sevcik, K.C., Johnson, M.J. Cycle Time Properties of the FDDI Token Ring Protocol. *IEEE Transactions on Software Engineering*, Vol. 13, No. 3, March 1987, pp. 376-385.
- [84] Special Issue on Token Rings. *IEEE Network*, January 1987.
- [85] Sprunt, B., Lehoczky, J., Sha, L. Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm. *Proceedings of the Real-Time Systems Symposium*, December 1988, pp. 251-258.
- [86] Stallings, W.S. *Local Networks: An Introduction*. New York: MacMillan Publishing Company, 1984
- [87] Stankovic, J.A. The Spring Architecture. *Proceedings of the Euromicro Workshop on Real-Time Systems*, June 1990.
- [88] Stankovic, J.A. Misconceptions about Real-Time Computing: A Serious Problem for Next-Generation Systems. *Computer*, October 1988, pp. 10-19.
- [89] Stankovic, J.A. A Perspective on Distributed Computer Systems. *IEEE Transactions on Computers*, Vol. C-33, No. 12, December 1984, pp. 1102-1115.
- [90] Stankovic, J.A., Ramamritham, K. What is Predictability for Real-Time Systems? *Real-Time Systems*, Vol. 2, No. 4, December 1990.
- [91] Stankovic, J.A., Ramamritham, K. The Spring Kernel: A New Paradigm for Real-Time Operating Systems. *ACM Operating Systems Review*, Vol. 23, No. 3, July 1989, pp. 54-71.
- [92] Stankovic, J.A., Ramamritham, K. *Tutorial Hard Real-Time Systems*. IEEE Computer Society Press, 1988.
- [93] Stankovic, J.A., Ramamritham, K., Cheng, S. Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems. *IEEE Transactions on Computers*, Vol. 34, No. 12, December 1985, pp. 1130-1143.
- [94] Strosnider, J.K. Highly Responsive Real-Time Token Rings. *Ph.D. Thesis*, Carnegie-Mellon University, Pittsburgh, PA, August 1988.
- [95] Strosnider, J.K., Marchok, T. Responsive, Deterministic IEEE 802.5 Token Ring Scheduling. *The Journal of Real-Time Systems*, Vol. 1, No. 2, September 1989, pp. 133-158.
- [96] Tanenbaum, A.S. *Computer Networks*. Englewood Cliffs, N.J.: Prentice-Hall, 1989.

- [97] Tobagi, F.A. Carrier Sense Multiple Access with Message-Based Priority Functions. *IEEE Transactions on Communications*, Vol. COM-30, January 1982, pp. 185-200.
- [98] Tokuda, H., Mercer, C.W., Ishikawa, Y., Marchok, T.E. Priority Inversions in Real-Time Communication. *Proceedings of the Real-Time Systems Symposium*, Santa Monica, CA, December 1989.
- [99] Towsley, D., Venkatesh, G. Window Random Access Protocols for Local Computer Networks. *IEEE Transactions on Computers*, Vol. C-31, No. 8, August 1982.
- [100] Valadier, J.C., Powell, D.R. On CSMA Protocols Allowing Bounded Channel Access Times. *IEEE International Conference on Distributed Computing Systems*, 1984.
- [101] Walter, C.J., Kieckhafer, R.M., Finn, A.M. MAFT: A Multicomputer Architecture for Fault-Tolerance in Real-Time Control Systems. *Proc. of the Real-Time Systems Symposium*, December 1985, pp. 133-140.
- [102] Weisbin, C.R., de Saussure, G., Einstein, J.R., Pin, F.G., Heer, E. Autonomous Mobile Robot Navigation and Learning. *Computer*, Vol. 22, No. 6, June 1989, pp. 29-35.
- [103] Weissler, R., Binder, R., Bressler, R., Rettberg, R., Walden, D. Synchronization and Multiple Access Protocols in the Initial Satellite IMP. *Fall COMP-CON*, September 1978.
- [104] Whittaker, W.L., Kanade, T. Japan Robotics Aim for Unmanned Space Exploration. *IEEE Spectrum*, Vol. 27, No. 12, December 1990, pp. 64-67.
- [105] Wensley, J.W., Lamport, L., Goldberg, J., Green, M., Levitt, K.N., Melliar-Smith, P.M., Shostak, R.E., Weinstock, C.B. SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control. *Proc. of the IEEE* 66(11), October 1978, pp. 1240-1255.
- [106] Wozencraft, J.M., Jacobs, I.M. *Principles of Communication Engineering*. John Wiley, 1987.
- [107] Zhao, W., Ramamritham, K. Virtual Time CSMA Protocols for Hard Real-Time Communication. *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 8, August 1987.
- [108] Zhao, W., Stankovic, J.A., Ramamritham, K. A Window Protocol for Transmission of Time Constrained Messages. *IEEE Transactions on Computers*, Vol. 38, No. 9, September 1990.