WINDOW MAC PROTOCOLS FOR REAL-TIME COMMUNICATION SERVICES

K. Arvind, K. Ramamritham, J.A. Stankovic
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

COINS Technical Report 90-127
January 1991

WINDOW MAC PROTOCOLS FOR REAL-TIME COMMUNICATION SERVICES*

K. ARVIND (arvind@cs.umass.edu)
KRITHI RAMAMRITHAM (krithi@cs.umass.edu)
JOHN A. STANKOVIC (stankovic@cs.umass.edu)

Department of Computer and Information Science
University of Massachusetts at Amherst
Amherst, MA 01003

COINS Technical Report 90-127
Submitted to IEEE Transactions on Communications

January 1991

Abstract

In this paper, we consider a distributed real-time system based on a local area network with a bus topology, and propose a new suite of real-time medium access control protocols. The proposed suite is devoid of deficiencies (in supporting real-time communication) found in existing standards for bus-based systems. The suite consists of five medium access control 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, and the protocol RTVC may be used to implement the abstraction of real-time virtual circuits. 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 derive various properties exhibited by the protocols and also present the results of a performance evaluation through simulation. The results of the simulation study show that, in addition to providing the advantage of structural homogeneity (enabling the use of a uniform medium access control logic and LAN controller hardware), the proposed suite of protocols also has excellent performance characteristics.

Index Terms: broadcast bus topology, connectionless service, connection-oriented service, distributed real-time system, local area network, medium access control protocol, multiaccess communication, priority resolution, real-time communication, window protocol.

*This work is part of the Spring Project at the University of Massachusetts funded in part by the Office of Naval Research under contract N00014-85-K-0398 and by the National Science Foundation under grant DCR-8500332 and CCR-8716858.

1 INTRODUCTION

A real-time system is a system whose correctness depends not only on the logical result of computation, but also on the time at which the results are produced ([36], [34]). *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 on different nodes is an important activity in a distributed real-time system. In this paper, 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.

In [2], 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.

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 [9] and the 802.4 standard [8]. 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 paper, 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 2), 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 6), on the worst case packet service times for the RTVC (Property 7), INTPVC (Property 9) and INTPDG window protocols (Property 10). 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 (explained in

Section 2) 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 paper also has the advantage of structural homogeneity, since all the protocols are based on a uniform window splitting paradigm.

The presentation in this paper is organized as follows. In Section 2, we develop the required context for the description of the medium access control protocols presented in this paper. We examine the communication requirements that arise in a distributed real-time system in Section 2.1. In Section 2.2, we present a brief description of RTLAN, focusing mainly on the logical link control layer. Section 2.2.1 contains a brief description of RTCOS and RTCLS. In Section 3, we present a homogeneous approach to MAC layer protocols based on a uniform window splitting paradigm. In Section 4 through Section 7, we employ this approach to develop the window protocol suite for supporting RTCOS and RTCLS. In Section 8, we present a brief survey of related work on MAC protocols. Section 9 contains a presentation of the results of a performance evaluation of the protocols. We conclude the paper in Section 10 with a brief summary.

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 fixed maximum delay constraints associated with them, is often termed real-time communication. However, in the rest of this paper 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.

2.1 *COMMUNICATION REQUIREMENTS*

Communication requirements in a distributed real-time system are induced by the need for interaction between various entities in the distributed system. Time-constrained 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.

Most current work in real-time communication assumes a open loop [22] model of a distributed real-time system in which the messages involved are mainly of two types, viz., *periodic* messages that typically carry sensor data, and *aperiodic* messages. Further both classes of messages are assumed to be statically specifiable. Periodic messages and a limited class of aperiodic messages (e.g., alarm messages) are usually classified as guarantee seeking, while the remaining aperiodic messages (e.g., certain status, control and advisory messages) are treated as best effort messages. However, such a simplistic model is likely to prove inadequate for the evolving next generation of closed loop autonomous real-time systems ([34],[35],[2]). Specialized network architectures that incorporate new classes of services and protocols will be required, in order to meet the more complex communication requirements of these dynamic systems. In [2], we have proposed RTLAN, a new local area network architecture, to support the communication requirements that arise in these systems. We provide a brief description of RTLAN in the next section.

2.2 *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 structured as a layered architecture (Figure 1) consisting of four layers, viz., 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 span 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

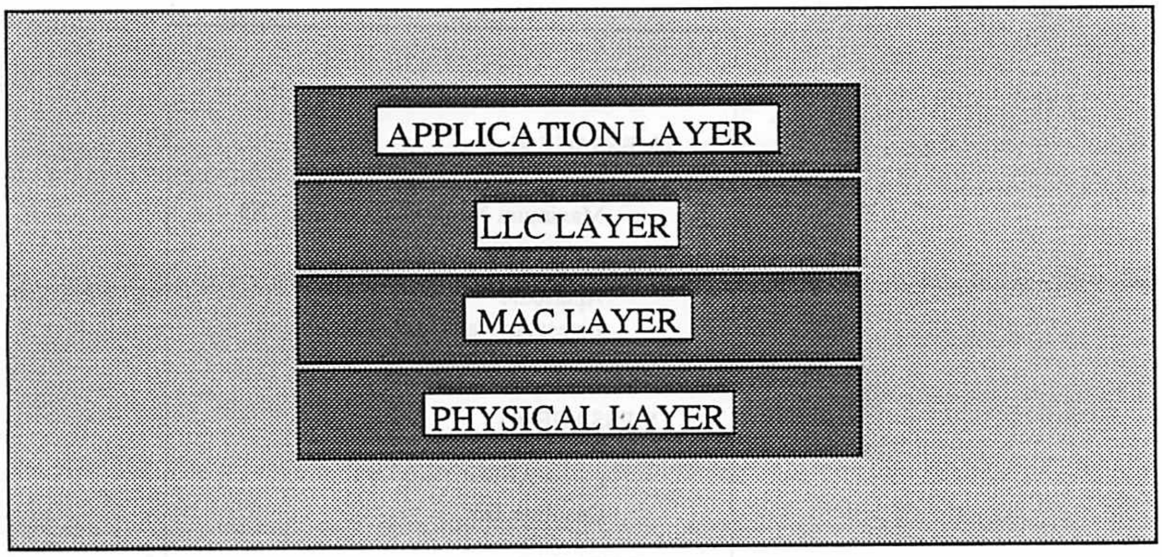


Figure 1: The RTLAN Architecture

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.

2.2.1 COMMUNICATION SERVICES

The logical link control layer of the RTLAN architecture provides new classes of real-time communication services to the application layer, that take the timing requirements of messages explicitly into account. The new service classes are known as *real-time connection-oriented service* (RTCOS) and *real-time connectionless service* (RTCLS).

RTCOS

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 connection*. A real-time connection 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,

RTC REQUIREMENTS

MESSAGE #1
- Periodic Period: **P1**
- Maximum Number of Cycles: **N1**

MESSAGE #2
- Aperiodic
- Latest Arrival Time **T2**
- Deadline **D2**

MESSAGE #3
- Aperiodic
- Latest Arrival Time **T3**
- Deadline **D3**

MESSAGE #4
- Aperiodic
- Latest Arrival Time **A4**
- Earliest Delivery Time **E4**
- Deadline **D4**

Figure 2: Real-Time Connection Requirements

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 connection is set up only if the specified requirements can be guaranteed; otherwise the sender is informed that the connection cannot be established.

The timing constraints specified may be fairly general and may include periodicity, arrival time, laxity, deadline, etc. Figure 2 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.

Two approaches that the LLC layer may use in order to provide such a guarantee are the ***priority assignment*** approach and the *real-time virtual circuit* approach. We discuss these below.

Priority Assignment Approach

The priority assignment approach may be used to handle the static periodic message components of systems that involve both static and dynamic communication requirements, by dedicating a separate physical channel for these components.

The approach is based on the *rate monotonic* priority assignment scheme [21]. 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 [21] 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

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

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 [18] have extended this result to the problem of scheduling n periodic messages on a shared bus. Strosnider (1988) 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. (1). Actual implementation of priority arbitration is left to suitable MAC protocols. Note that neither of the two standards proposed for bus-based systems support priority resolution. The IEEE 802.3 CSMA/CD protocol has no notion of packet priorities. The IEEE 802.4 token bus protocol does support up to 4 priority classes, corresponding to 4 different classes of traffic. However, it does not support packet level priority-based arbitration.

Real-Time Virtual Circuit Approach

The real-time virtual circuit (RTVC) approach may be used to guarantee the timing requirements of both statically known and dynamically arising messages. 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. The packet service time consists of two components, the physical *channel access time* and the packet *transmission time*. 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

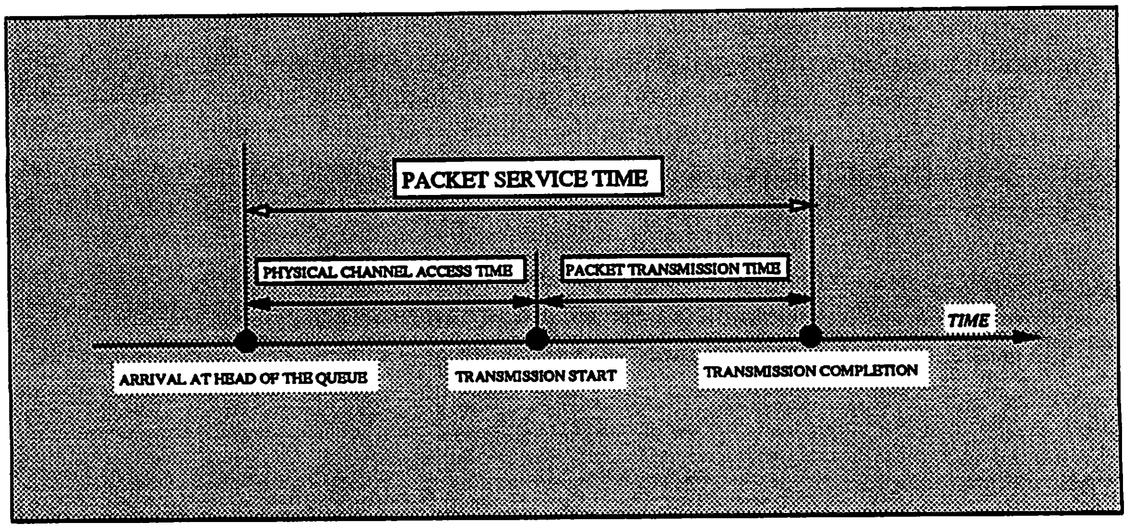


Figure 3: Packet Service Time

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 can be unbounded. 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 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* [5] 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) discipline[1]) 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 the following 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 illustrates these ideas. In this example, there is one RTVC (with a worst case service time of 10) which already has three guaranteed packets waiting to be

[1]A commonly used discipline in the scheduling of real-time tasks is the *minimum laxity 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 laxity first policy can do so.*

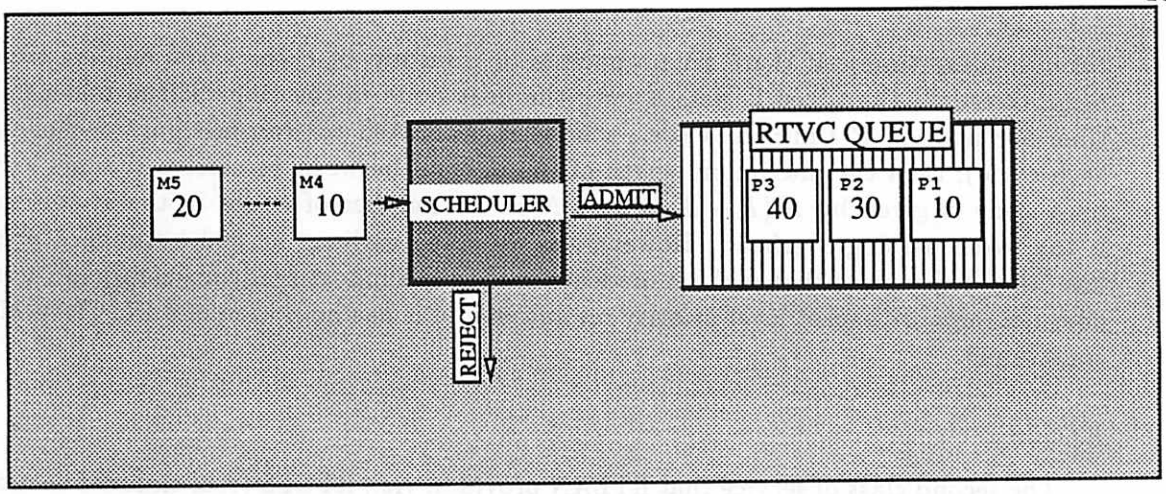


Figure 4: Real-Time Virtual Circuit Scheduling

transmitted. Packet P1 has a laxity (time until deadline for start of transmission) of 10, P2 has a laxity of 30 and P3 has a laxity of 40. If a new request M4 consisting of a packet of laxity 10 arrives, then it cannot be accepted by the system, if the MLF discipline is used, since the deadlines of both M1 and M4 cannot be simultaneously met. However a request M5 consisting a packet 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.

RTVCs are abstractions provided by the MAC layer to the LLC layer. Protocols in the MAC layer transform the underlying raw multiple access shared channel with potentially unbounded times, into logical channels with bounded packet service times, viz., RTVCs. The MAC layer has to employ suitable protocols in order to provide such an abstraction.

For instance, the 802.3 CSMA/CD standard protocol cannot be used to implement real-time virtual circuits. This is a random access protocol in which, at the beginning of a transmission epoch, every node that has a packet awaiting transmission transmits the packet. If more than one node has a packet to transmit, a collision results. In this case, each node is *randomly* assigned one of the next 2^k slots for transmitting its packet, where k is the number of collisions that have occurred in the current transmission epoch. This collision resolution strategy, known as *binary exponential backoff* may result in a packet never being transmitted. Thus this protocol does not guarantee bounded channel access times.

The 802.4 standard token-passing bus protocol may however be used to implement real-time virtual circuits. In this protocol, the nodes are organized in a logical ring. A 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 $N(P + \beta)$, where N is the number of nodes in the system and β is the token passing overhead. However, as we noted earlier, the 802.4 standard protocol has other deficiencies that make it inadequate to fully support real-time communication. It does not support packet level priority resolution and it has no notion of packet level timing constraints.

RTCLS

The second class of service that RTLAN provides, viz., RTCLS (real-time connectionless service), is an unreliable *connectionless* service 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 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. The individual packets that are transmitted under RTCLS are referred to as *real-time datagrams*.

In order to support RTCLS, the LLC layer requires support from the MAC layer in the form of real-time MAC protocols that explicitly consider timing constraints of packets in arbitrating access to the shared bus. Examples of such real-time protocols are the VTCSMA protocol [42] and the MLF window protocol [43], which try to order the transmission of messages according to a minimum laxity first policy. Note that neither of the two standard MAC protocols for local area networks, viz, the 802.3 CSMA/CD standard and the 802.4 token bus standard, have any notion of timing constraints of individual packets. They do not consider the timing requirements of packets in arbitrating access to the channel. Thus they are not well suited for providing a best-effort service.

With this, we conclude the presentation of the framework within which the rest of the paper fits. In the next section, we introduce the main contribution of this paper, a new homogeneous suite of MAC protocols for real-time communication that is capable of supporting both RTCOS and RTCLS.

3 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 [43]. 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 [43] 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.

3.1 *WINDOW SPLITTING PROCEDURE*

Window protocols and other members of the splitting algorithm family like stack and tree protocols (e.g., [3], [4], [16], [19], [40]) 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 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, 90 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 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.

3.1.1 PSEUDOCODE

Figure 6 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

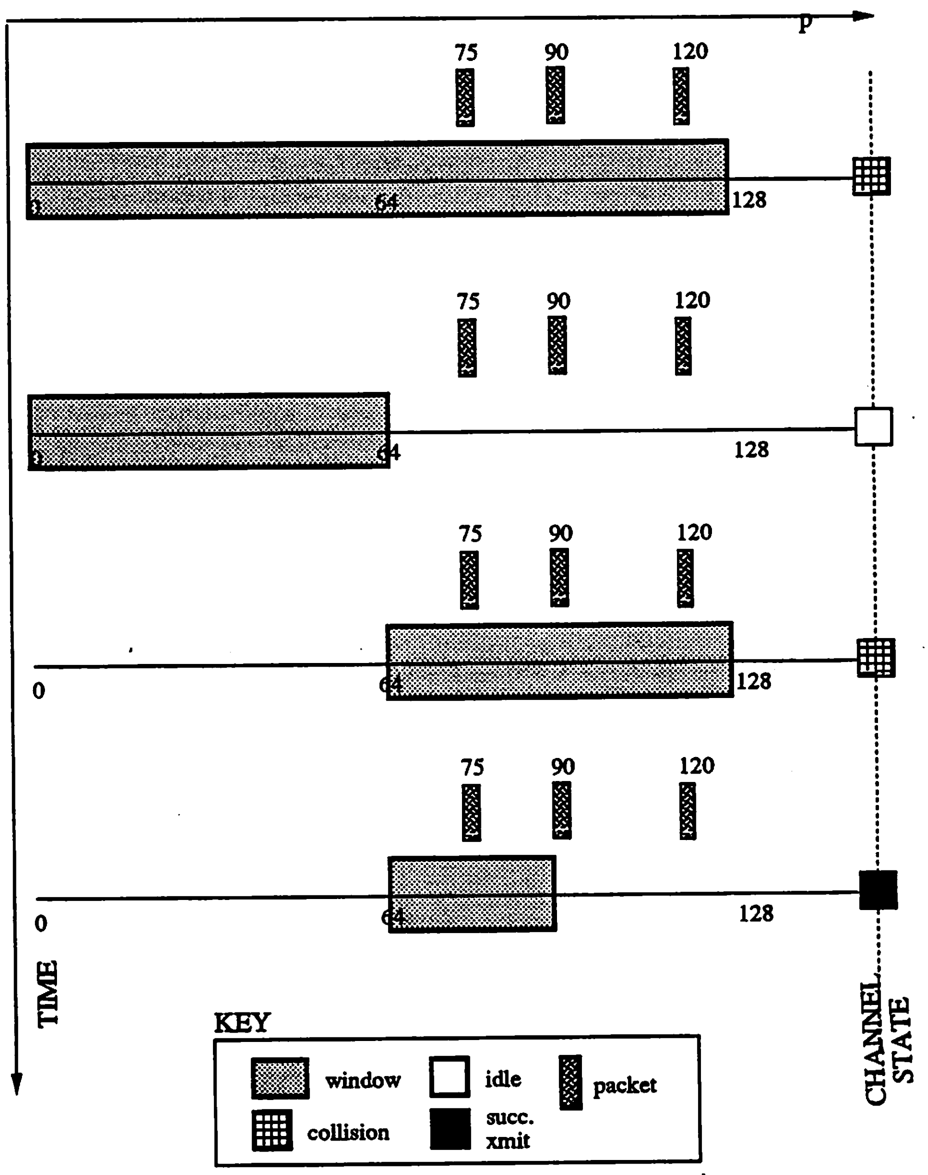


Figure 5: An example of the window procedure in operation

```
1.  window ($\Pi, \Delta, R$)
2.    begin
3.        $\delta \leftarrow \delta_{max} \leftarrow$ ceil2 ($\Delta$); $\pi \leftarrow 0$; $\sigma \leftarrow$ 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.                       $\sigma \leftarrow$ Left;
21.                       $\delta \leftarrow \frac{\delta}{2}$;
22.                       if ($\delta < 1$) then window($\Omega, A, adr$);
23.                       break;
24.                   case idle:
25.                       if $\sigma$ = Right then return;
26.                       $\sigma \leftarrow$ Right;
27.                       $\pi \leftarrow \pi + \delta$;
28.                       break;
29.               end switch
30.       end forever;
31.   end window.
```

Figure 6: The window splitting procedure

parameter space under consideration. The argument $\Delta \leq |\Pi|$ is used to compute[2] (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.

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

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 binary[3] 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 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.

3.1.2 PROPERTIES OF window

PROPERTY 1 *The procedure* window *implements PBCR.*

<u>**PROOF**</u>: 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 2 *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:*

$$\xi_{max} \leq 2\log_2 \lceil \Delta \rceil - 1, \tag{2}$$

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

<u>**PROOF**</u>: Initially, let us assume that Δ is a power of 2. Let

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

[3] A procedure similar to window is specified in the IEEE 802.4 token bus standard for station *addition*. However this protocol uses recursive quarternary partitioning.

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 $\delta_{max} = \text{ceil2}(\Delta) = 2$ (line 3 of Figure 6) 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. (2).
Induction Step: Assume Eq. (2) 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. (2) 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 (2). **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:*

$$\xi_{max} \le 2\left(\lceil \log_2 \Delta \rceil + \lceil \log_2 A \rceil\right) - 2, \tag{3}$$

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 2, except that the worst case contention overhead incurred in order to resolve a tie is included.

PROPERTY 3 *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 ξ is given by:*

```
1.  forever do
2.      window($\Phi$, K, pri);
3.  end forever
```

Figure 7: The PRI Window Protocol

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 2, 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.

4 *THE PRI WINDOW PROTOCOL*

Figure 7 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 4 *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 1 and the definition of "pri".

In Section 8.1, we will see that the PRI window protocol has a smaller average contention resolution overhead (Property 3) 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.

5 *THE RTDG WINDOW PROTOCOL*

The pseudocode for the RTDG window protocol is shown in Figure 8. This protocol repeatedly invokes the window contention resolution procedure with the space of possible

```
1.  forever do
2.      window($\Lambda, L$, mlf);
3.  end forever
```

Figure 8: The RTDG Window Protocol

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.

5.1 *PROPERTIES OF RTDG*

PROPERTY 5 *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 1 and the definition of rule "mlf".

Property 5 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 6 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 3) is small. This may be seen from the following property and example.

PROPERTY 6 *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 given by:*

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

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 a packet that arrives during a contention resolution interval 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 3 and the uniform distribution assumption, if we make the worst case assumption that that $\Pr[E] = 1$, i.e., a packet that arrives 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 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 $\alpha = 0.01$).

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

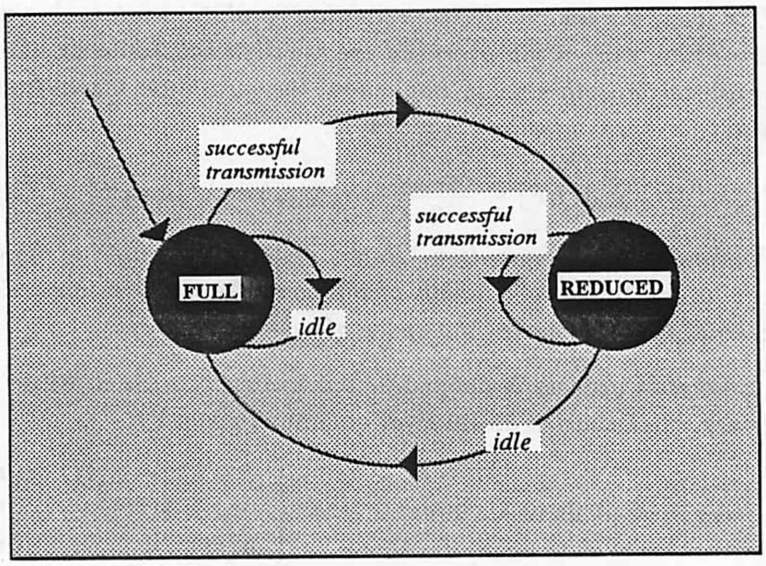


Figure 9: RTVC Protocol State Machine

In Section 9, 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.

6 *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

```
1.  $C \leftarrow$ 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 10: The RTVC window protocol

instant is denoted Γ and is referred to as the current space.

Figure 9 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, 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.

6.1 PSEUDOCODE

Figure 10 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.

6.2 *PROPERTIES OF RTVC*

PROPERTY 7 *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 2.*

<u>PROOF</u>: 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 2.

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 10 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 7, 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 8 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 unspecified protocol PROT for which the worst case service time is given by $N_{rtvc}(P + \beta)$ (See Section 8.1.1). 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 8 *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 2, 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 9. 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.

7 *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 [16]. 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 [8] and the FDDI protocol ([29], [30]) 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 $2 * \text{TTRT}$ [32]. If each node is permitted to transmit a maximum of one synchronous packet per token rotation then $\text{TTRT} = N * P$, 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* [27]. 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 ([33], [20]). 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.

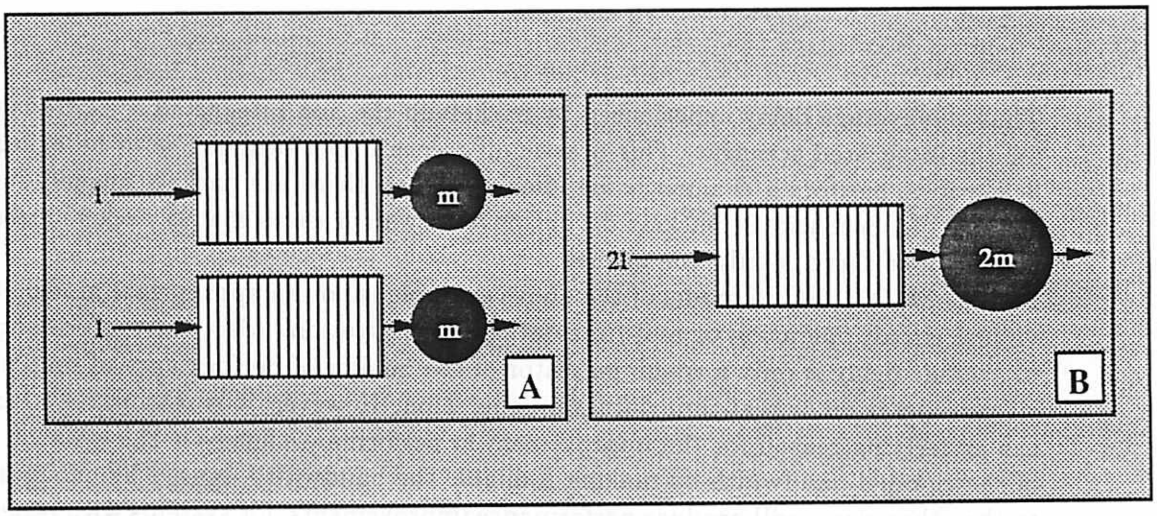


Figure 11: Two Queueing Scenarios

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 11 illustrates two ways of handling the two classes of traffic. In scenario A, each stream of traffic has a server of capacity $m = \frac{1}{s}$ 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,

$$W = E[X] + \frac{\lambda E[X^2]}{2(1-\rho)}, \tag{4}$$

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

$$W = S\left(1 + \frac{\rho}{2(1-\rho)}\right) = K_\rho S, \tag{5}$$

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 commin bus. We have developed two integrated protocols, INTPVC and INTPDG in Section 7.1 and Section 7.2 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 the token bus and FDDI protocols (Section 7.1); 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 7.2).

7.1 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 12 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 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.

7.1.1 PSEUDOCODE

Figure 13 contains the pseudocode for this protocol. The code essentially describes the operations of the state machine depicted in Figure 12 and is self-explanatory. Note

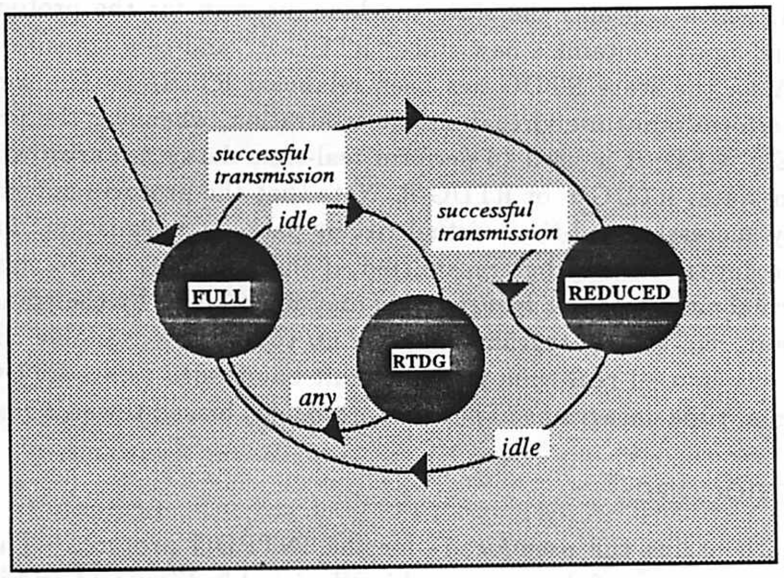


Figure 12: INTPVC Protocol State Machine

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).

7.1.2 PROPERTIES OF INTPVC

PROPERTY 9 *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.

<u>PROOF</u>: This result follows from Property 7, 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.

7.2 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 14, which contains the state machine diagram for the protocol. Compared to Figure 12, 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.

7.2.1 PSEUDOCODE

Figure 15 contains the pseudocode for the INTPDG protocol. The code essentially describes the operations of the state machine depicted in Figure 14 and is self-explanatory.

7.2.2 PROPERTIES OF INTPDG

PROPERTY 10 *An upper bound on the service time S of a real-time virtual circuit packet is given by*

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

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

PROOF: 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 12. The result then follows from Property 9. **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.

With this we conclude our description of the window protocols. In the next section, we examine various other non-standard protocols that have been proposed in the literature, that may be used to support RTCOS and RTCLS. Later in Section 9.2, we abstract out the essential elements of these protocols to develop suitable baseline protocols that represent idealizations of these protocols. We use these baseline protocols in an evaluation of the performance of the window protocols proposed in this paper.

8 RELATED WORK

8.1 PRIORITY RESOLUTION PROTOCOLS

Priority resolution protocols for *bus-based* systems are classified by Valadier and Powell [41] into three categories, on the basis of the underlying scheme involved. In the first scheme known as the *deference delay* scheme, priority arbitration is accomplished by requiring a node to defer its transmission by amount that is proportional to its priority value. The protocols proposed in [7] and [39] are examples of this scheme. The second scheme known as the *preamble length* scheme implements priority arbitration by requiring each node to transmit a preamble signal that is proportional to its priority. The protocol proposed in [11] is an example of a protocol that employs this scheme. The average priority resolution overhead for both these protocols is a linear function of the number of priority levels. The third scheme known as the *forcing headers* scheme requires each node to transmit sequentially, the bit string that defines the node priority. The scheme makes use of the inherent wired-OR property of a broadcast bus to implement priority arbitration. The average priority resolution overhead for this scheme increases logarithmically with the number of priority levels (K) and is given by $\lceil \log_2 K \rceil$ round trip propagation delays. The MLMA or multilevel multiaccess protocol [31] 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. The PRI window protocol that we proposed in Section 4 has an average overhead ($= \lceil \log_2 K \rceil - 0.5$) that is slightly smaller than that for the forcing headers scheme, and is thus better than all the above schemes, in this regard.

The IEEE 802.5 standard for token *ring* local area networks supports a priority resolution scheme, that makes use of two special fields in the token known as the reservation field and the priority field. Strosnider and Marchok discuss the use of the token ring priority resolution scheme for scheduling statically specified message sets in detail, in [38]. However this scheme cannot be used on a bus-based system (e.g., token passing bus), since it requires that each transmitted message circulate around the ring. In the token-passing bus protocol, only the token *circulates* around the ring; regular messages are merely *broadcast* on the bus.

8.1.1 REAL-TIME VIRTUAL CIRCUIT PROTOCOLS

Examples of MAC protocols that can guarantee a bounded channel access time are TDMA, virtual token passing protocols, the 802.3D protocol and the waiting room protocol.

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 $S * P$, 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 $S * P$.

Virtual token-passing protocols are conceptually similar to the token passing bus. The nodes in the system are organized in a logical ring as in the token bus protocol. However, a real token is not transmitted in order to transfer channel access rights from a node to its neighbor. Completion of transmission of the permitted maximum number of packets or the absence of transmission (e.g., when there are no packets to transmit) by the node that currently has permission to transmit, constitutes a signal that implicitly transfers channel access rights to the node's forward neighbor. If 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, P is the time required to transmit a single packet and τ is the end-to-end propagation delay. BRAM [6] and MSAP [14] are examples of virtual token-passing protocols.

The 802.3D protocol proposed by Le Lann [19] 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 standard for the ethernet which, owing to the collision resolution strategy (binary exponential backoff) that it (ethernet) uses, 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 τ is the round trip propagation delay.

The notion of waiting room priorities ([41], [26]) 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 [41].

The worst case channel access times for the various non-standard protocols that we have described above (except the waiting room protocol that has a larger worst case channel access time) may be represented by $N\,(P+\beta)$, where N is the number of real-time virtual circuits in the system, P is the packet transmission time and β is the per packet contention resolution overhead. For example, β is zero for TDMA and is equal to the end-to-end propagation delay for the virtual token passing protocols. For the RTVC window protocol, $\beta = \xi_{max}$, which is slightly larger than that for the other protocols. However, we noted in the remark after Property 8 that, in spite of the slightly larger value of β, the performance of the RTVC window protocol is close to that of an idealized protocol with $\beta = 0$.

3.2 RTCLS/MAC PROTOCOLS

In Section 2.2.1, 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 [17] 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 discards 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 [12] have considered the last-in first-out (LIFO) policy, and shown that under certain conditions (when the deadlines are i.i.d. with concave cumulative distribution functions) this policy is optimal. In [25], 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 [42] and the MLF window protocol [43] 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 [42], 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 choosing different message parameter axes along which the virtual clock is run. For example, choice of message arrival time as the parameter used by the protocol corresponds to the first-in first-out transmission policy [24], while use of message laxity corresponds to the minimum laxity first policy.

The MLF window protocol belongs to the class of inference-seeking protocols [43]. 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. 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 [43]. An 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.

9 PERFORMANCE EVALUATION

In this section, we describe a simulation study conducted in order to evaluate the performance of the window protocols presented in this paper. In Section 9.1, we define the performance metrics that are appropriate for evaluating the performance of the MAC protocols considered in this paper. In Section 9.2, we present a set of idealized MAC protocols that serve as baselines for comparison. In Section 9.3, we describe the model of the system that the simulation programs assume. In Section 9.4, we present the results of the simulation study.

9.1 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 [13], 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* ([16],[42],[43]). The loss fraction (LF) is defined as:

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

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 [5] for the measurement of the quality of performance of dynamic task scheduling algorithms. The guarantee ratio is defined as

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

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).

9.2 *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 8.1.1, 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.

9.2.1 *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.

9.2.2 *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.

9.2.3 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 8.1.1 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.

9.2.4 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 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),

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

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.

9.3 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 μseconds 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

$$\rho_{dg} = \frac{N\lambda_{dg}}{\mu}. \tag{6}$$

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
$$\rho_{vc} = \frac{N_{rtvc}\lambda_{vc}}{\mu}. \tag{7}$$
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.

9.4 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 9.2. 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.

In order to keep this presentation short, we have provided only representative plots, even though our experiments were more comprehensive than what is presented here. 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.

9.4.1 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 16 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, based on the minimum laxity first policy, 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.

9.4.2 *RTVC Window Protocol*

Figure 17 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. 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. This is illustrated by the plots for the TDMA protocol. When the average laxity is close to the worst case channel access time for TDMA (2500 time units in the plots), the performance of the TDMA protocol is slightly better than that of the RTVC window protocol (whose worst case channel access time is 2725 units). Note that under certain conditions, the performance of TDMA is actually poorer than that of the RTVC window protocol. This happens in Figure 17, for example, when $\rho_{vc} = 1$ and the average laxity is between approximately 3000 and 30000 time units, and when $\rho_{vc} = 1.5$ and the average laxity is between 3000 and 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.

9.4.3 *Integrated Window Protocols*

Figure 18 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 a higher loss fraction than VCTIMER and INTPDG has a higher guarantee ratio than VCTIMER.

9.4.4 *Effects of Integration*

Figure 19 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 RTDG and RTVC services up to a certain load (which depends on the RTDG load). Beyond this value, the quality improvement in RTVC services is maintained, but at the expense of the RTDG 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 tradeoff the quality of RTDG services at higher RTVC 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) for a wide range of real-time virtual circuit loads ($\rho_{vc} \leq 1$), 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 tradeoff the quality of service for real-time virtual circuit packets to some extent.

10 CONCLUSION

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, and closely approximates an ideal baseline protocol that implements the same function. The performance of RTDG is also superior to that of existing standards that have no notion of packet timing constraints. The protocol RTVC implements real-time virtual circuit arbitration and was shown to closely approximate an idealized baseline protocol that implements the same function. 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. The INTPVC window protocol provides a better quality of service for real-time virtual circuit packets than an idealized baseline protocol known as VCTIMER. In addition, 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. The INTPDG window protocol provides a better quality of service for real-time datagram packets than the VCTIMER baseline protocol. 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 2), the probability that the RTDG window protocol fails to implement the minimum laxity first policy exactly (Property 6), the worst case packet service times for the RTVC window protocol (Property 7), INTPVC window protocol (Property 9) and the INTPDG window protocol (Property 10). We also presented results of a simulation study conducted in order to evaluate the performance of the protocols.

In addition to overcoming the deficiencies of existing standards, and to the good performance characteristics, 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.

```
$\Gamma \leftarrow \Gamma_0$; state $\leftarrow$ FULL;
forever do
    switch (state)
        case FULL:
            window($\Gamma, C,$ cap);
            event $\leftarrow$ *successful transmission* ? *idle*;
            switch (event)
                case *successful transmission*:
                    t $\leftarrow$ *RTVC whose packet was transmitted*;
                    foreach c $\in \Gamma$
                        if ($c \leq t$) then $\Gamma \leftarrow \Gamma - \{c\}$;
                    end foreach;
                    state $\leftarrow$ REDUCED;
                    break;
                case *idle*:
                    state $\leftarrow$ RTDG;
                    break;
            end switch;
        case REDUCED:
            window($\Gamma, C,$ cap);
            event $\leftarrow$ *successful transmission* ? *idle*;
            switch (event)
                case *successful transmission*:
                    t $\leftarrow$ *RTVC whose packet was transmitted*;
                    foreach c $\in \Gamma$
                        if ($c \leq t$) then $\Gamma \leftarrow \Gamma - \{c\}$;
                    end foreach;
                    state $\leftarrow$ REDUCED;
                    break;
                case *idle*:
                    $\Gamma \leftarrow \Gamma_0$; state $\leftarrow$ FULL;
                    break;
            end switch;
        case RTDG:
            window($\Lambda, L,$ mlf);
            state $\leftarrow$ FULL;
            break;
    end switch;
end forever
```

Figure 13: The INTPVC window protocol

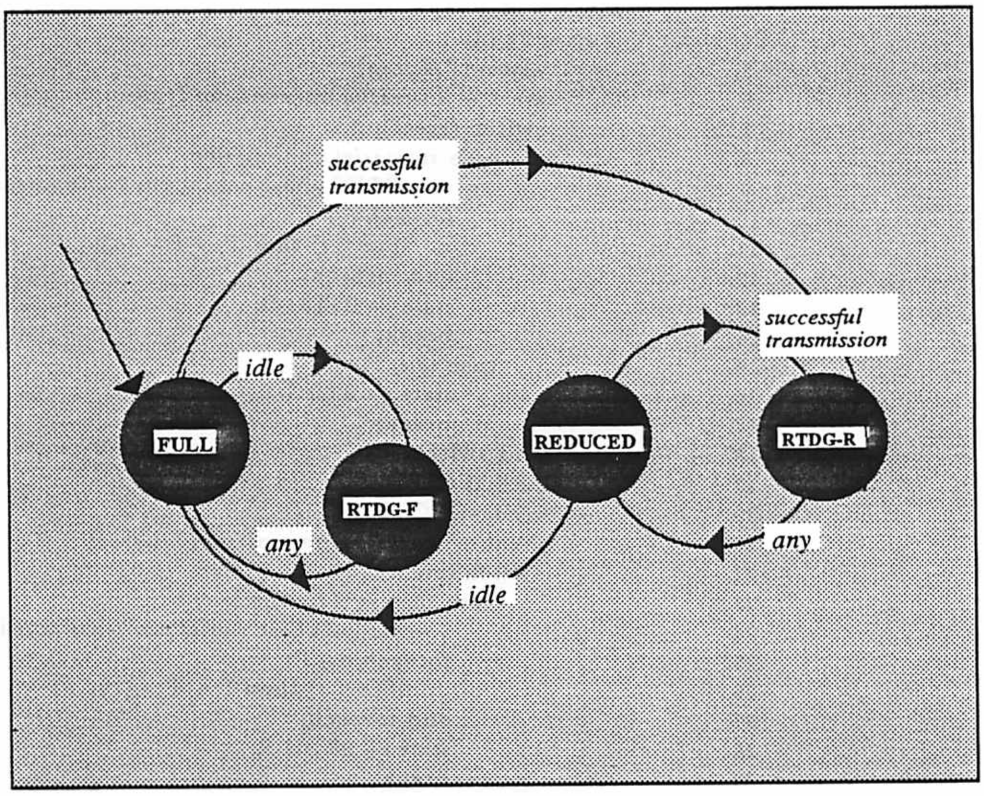
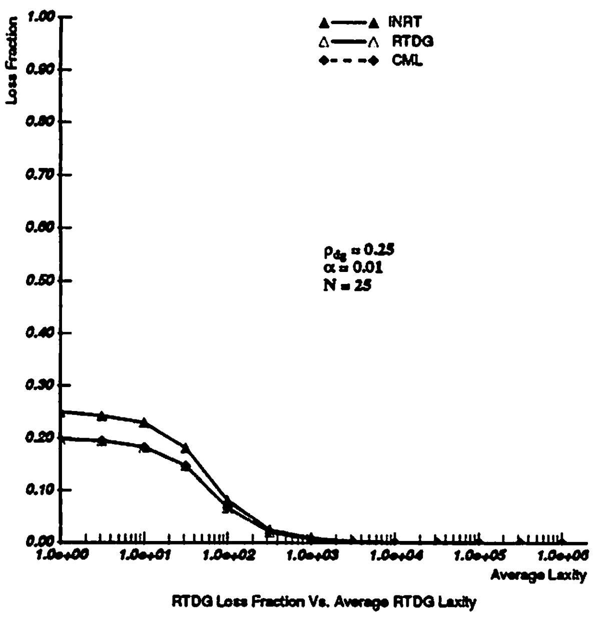
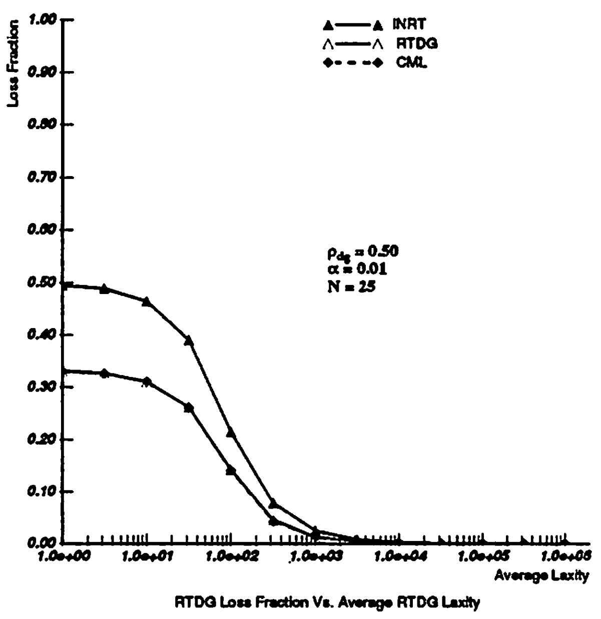


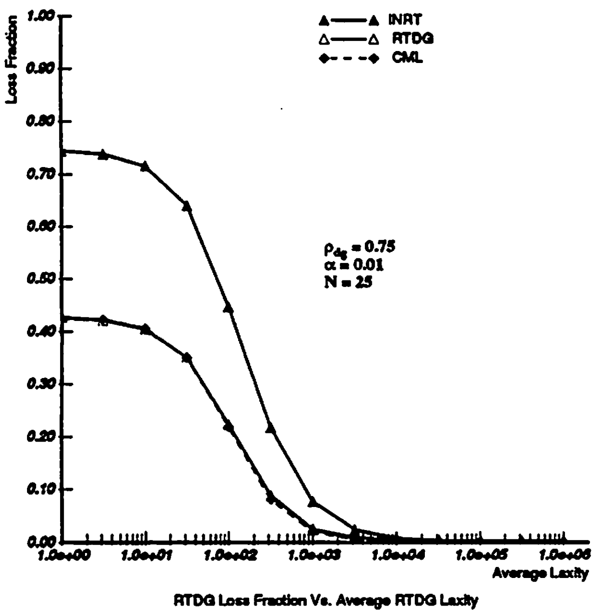
Figure 14: INTPDG Protocol State Machine

```
$\Gamma \leftarrow \Gamma_0$; state $\leftarrow$ FULL;
forever do
    switch (state)
        case FULL:
            window($\Gamma$, C, cap);
            event $\leftarrow$ *successful transmission* ? *idle*;
            switch (event)
                case *successful transmission*:
                    t $\leftarrow$ *RTVC whose packet was transmitted*;
                    foreach c $\in \Gamma$
                        if ($c \leq t$) then $\Gamma \leftarrow \Gamma - \{c\}$;
                    end foreach;
                    state $\leftarrow$ RTDG-R;
                    break;
                case *idle*:
                    state $\leftarrow$ RTDG-F;
                    break;
            end switch;
        case REDUCED:
            window($\Gamma$, C, cap);
            event $\leftarrow$ *successful transmission* ? *idle*;
            switch (event)
                case *successful transmission*:
                    t $\leftarrow$ *RTVC whose packet was transmitted*;
                    foreach c $\in \Gamma$
                        if ($c \leq t$) then $\Gamma \leftarrow \Gamma - \{c\}$;
                    end foreach;
                    state $\leftarrow$ RTDG-R;
                    break;
                case *idle*:
                    $\Gamma \leftarrow \Gamma_0$; state $\leftarrow$ FULL;
                    break;
            end switch;
        case RTDG-F:
            window($\Lambda$, L, mlf);
            state $\leftarrow$ FULL;
            break;
        case RTDG-R:
            window($\Lambda$, L, mlf);
            state $\leftarrow$ REDUCED;
            break;
    end switch;
end forever
```

Figure 15: The INTPDG window protocol







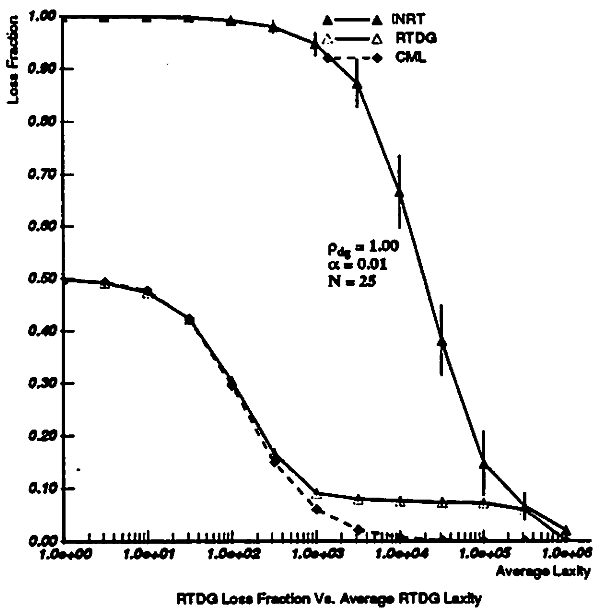


Figure 16: Simulation study results: RTDG Protocol

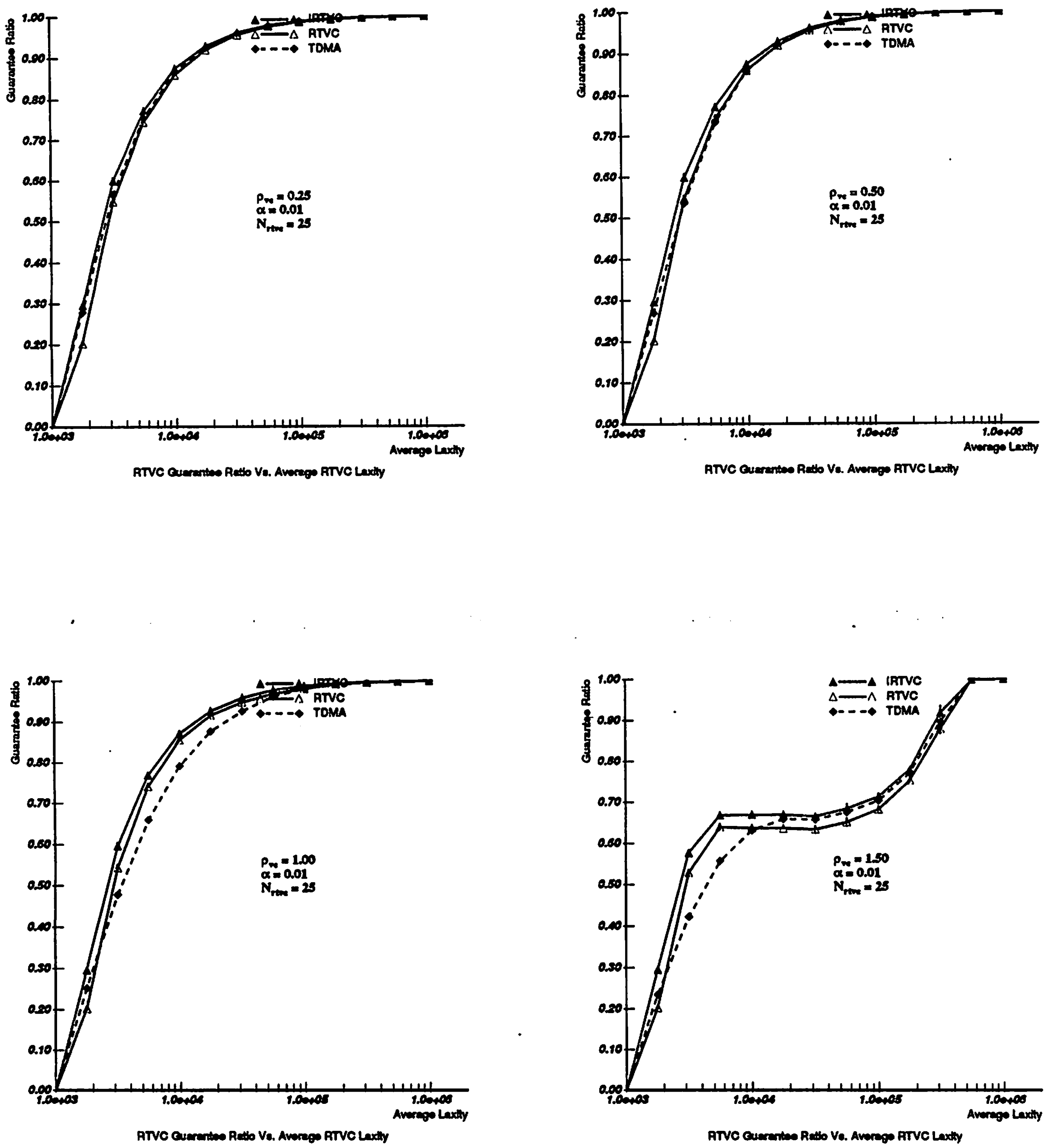
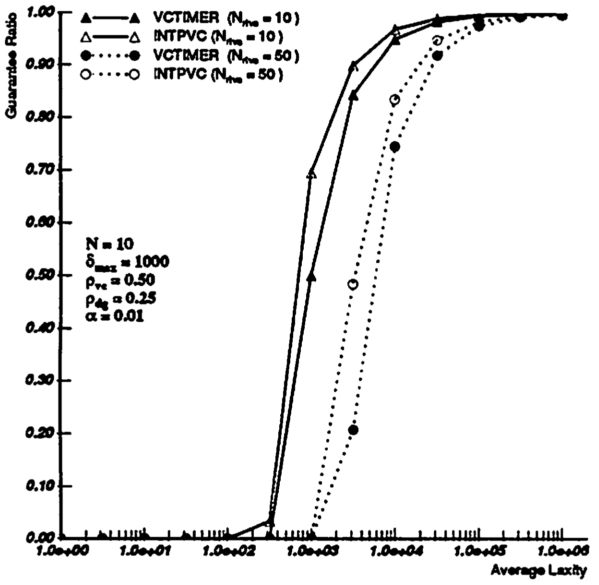
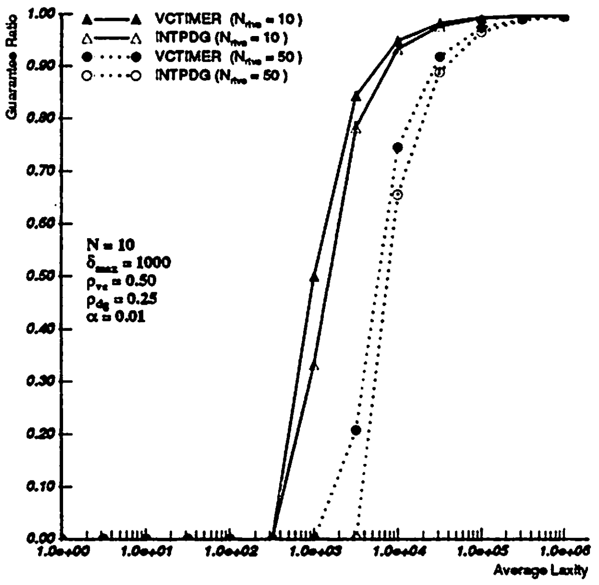


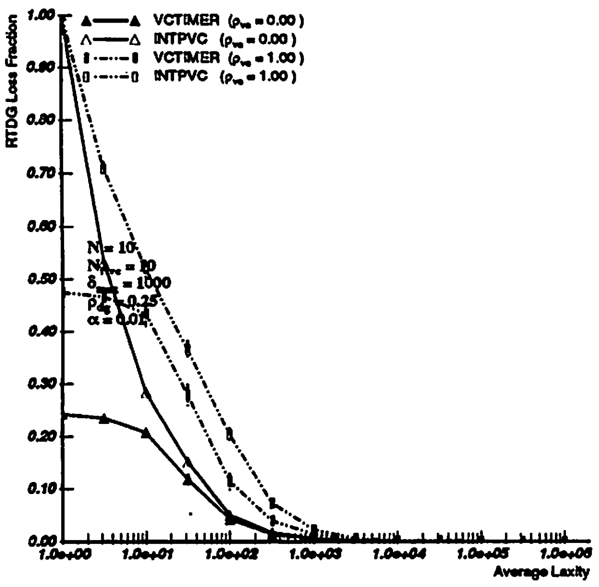
Figure 17: Simulation study results: RTVC protocol



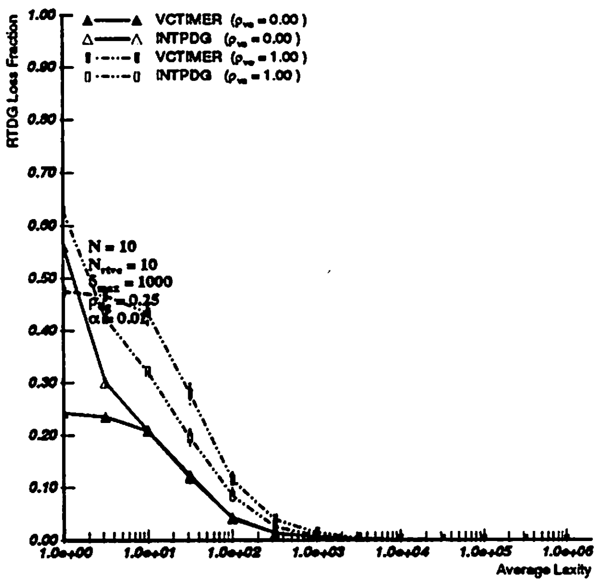
Comparison of RTVC Guarantee Ratio for VCTIMER and INTPVC



Comparison of RTVC Guarantee Ratio for VCTIMER and INTPDG



Comparison of RTDG Loss Fraction for VCTIMER and INTPVC



Comparison of RTDG Loss Fraction for VCTIMER and INTPDG

Figure 18: Simulation study results: INTPVC and INTPDG

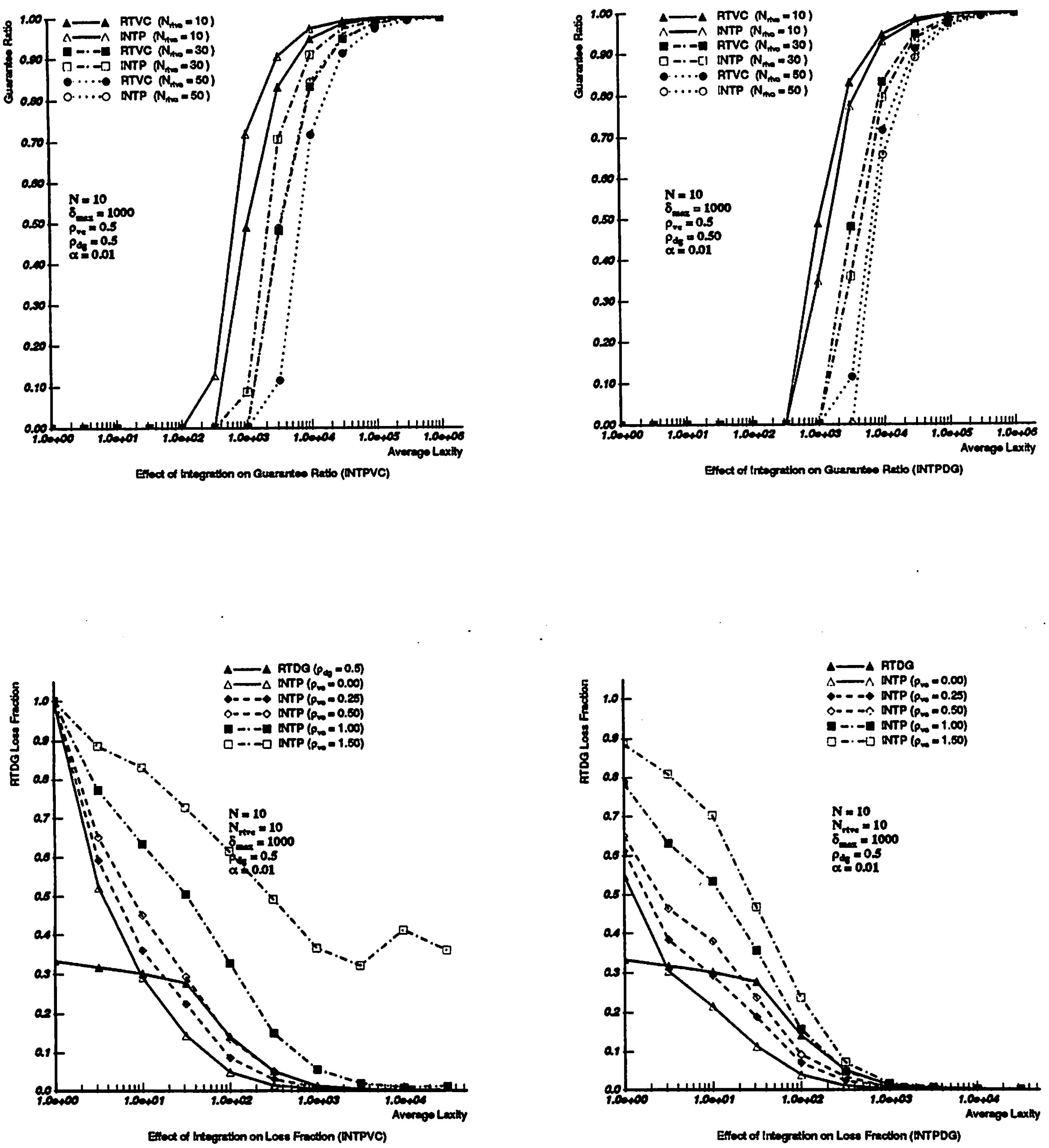


Figure 19: Simulation study results: Effects of Integration

BIBLIOGRAPHY

[1] Abramson, N.,The Aloha System — Another Alternative for Computer Communications, *Proceedings of the AFIPS Fall Joint Computer Conference*, **37**, Fall 1970.

[2] 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.

[3] Bertsekas, D., Gallager, R., *Data Networks*, Prentice Hall Inc., Englewood Cliffs, NJ, 1987.

[4] Capetanakis, J.I., Generalized TDMA: The Multi-Accessing Tree Protocol, *IEEE Transactions on Communications*, Vol. COM-27, No. 10, October 1979.

[5] 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.

[6] Chlamtac, I., Franta, W.R., Levin, D., BRAM: The Broadcast Recognizing Access Method, *IEEE Transactions on Communications*, Vol. COM-27, No. 10, October 1979.

[7] Franta, W.R., Bilodeau, M.B., Analysis of a Prioritized CSMA Protocol Based on Staggered Delays, *Acta Informatica*, Vol. 13, 1980, pp. 299-324.

[8] *IEEE Standards for Local Area Networks. Token-Passing Bus Access Method and Physical Layer Specifications*, IEEE, 1982.

[9] *IEEE Standards for Local Area Networks: Carrier Sense Multiple Access with Collision Detection*, IEEE, New York, 1985.

[10] *IEEE Standards for Local Area Networks. Token Ring Access Method*, IEEE, 1986.

[11] 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.

[12] 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)*, pp. 1073-1074, Tampa, Florida, 1989.

[13] Kleinrock, L., *Queueing Systems*, Vol. I, 1976.

[14] 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.

[15] 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.

[16] Kurose, J.F., Schwartz, M., Yemini, Y., Multiple-Access Protocols and Time-Constrained Communication, *Computing Surveys* 16(1):43-70, March 1984.

[17] 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.

[18] Lehoczky, J.P., Sha, L., Performance of Real-Time Bus Scheduling Algorithms, *ACM Performance Evaluation Review*, Special Issue 14(1), May 1986.

[19] 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.

[20] Le Lann, G., Real-Time Protocols, *Local Area Networks, An Advanced Course*, Hutchison, D., et al, Editors, Springer Verlag, 1983.

[21] 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.

[22] Martin, J., *Design of Real-Time Computer Systems*, Prentice-Hall, Englewood Cliffs, 1967.

[23] 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.

[24] 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.

[25] 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.

[26] Ramamritham, K. Channel Characteristics in Local-Area Hard Real-Time Systems, *Computer Networks and ISDN Systems*, Vol 13, 1987, pp. 3-13.

[27] 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.

[28] 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.

[29] Ross, F.E., FDDI - a Tutorial, *IEEE Communications Magazine*, Vol. 24, No. 5, May 1986.

[30] 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.

[31] Rothauser, E.H., Wild, D., MLMA-A Collision-Free Multi-Access Method, *Proceedings IFIP Congress 77*, 1977.

[32] 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.

[33] Stankovic, J.A., A Perspective on Distributed Computer Systems, *IEEE Transactions on Computers*, Vol. C-33, No. 12, December 1984, pp. 1102-1115.

[34] Stankovic, J.A., Misconceptions about Real-Time Computing: A Serious Problem for Next-Generation Systems, *Computer*, October 1988, pp. 10-19.

[35] Stankovic, J.A., Ramamritham, K., What is Predictability for Real-Time Systems?, *Real-Time Systems*, Vol. 2, No. 4, December 1990.

[36] Stankovic, J.A., Ramamritham, K., *Tutorial Hard Real-Time Systems*, IEEE Computer Society Press, 1988.

[37] Strosnider, J.K., Highly Responsive Real-Time Token Rings, *Ph.D. Thesis*, Carnegie-Mellon University, Pittsburgh, PA, August 1988.

[38] 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.

[39] Tobagi, F.A., Carrier Sense Multiple Access with Message-Based Priority Functions, *IEEE Transactions on Communications*, Vol. COM-30, January 1982, pp. 185-200.

[40] Towsley, D., Venkatesh, G., Window Random Access Protocols for Local Computer Networks, *IEEE Transactions on Computers*, Vol. C-31, No. 8, August 1982.

[41] Valadier, J.C., Powell, D.R., On CSMA Protocols Allowing Bounded Channel Access Times, *Fourth International Conference on Distributed Computing Systems*, San Francisco, May 1984.

[42] 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.

[43] Zhao, W., Stankovic, J., Ramamritham, K., A Window Protocol for Transmission of Time Constrained Messages, *IEEE Transactions on Computers*, September 1990.