A Local Area Network Architecture for Communication in Distributed Real-Time Systems

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

COINS Technical Report 91-04
January 1991

A LOCAL AREA NETWORK ARCHITECTURE FOR COMMUNICATION IN DISTRIBUTED REAL-TIME SYSTEMS*

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

To Appear in
REAL-TIME SYSTEMS, VOL. 3, NO. 2, MAY 1991

January 1991

Abstract

Distributed real-time systems of the future will require specialized network architectures that incorporate new classes of services and protocols in order to support time-constrained communication. In this paper, we propose a new local area network architecture for such systems. This four layered architecture is characterized by new classes of connection-oriented and connectionless services that take into account the timing constraints of messages. We describe various aspects of the logical link control layer of the architecture and various real-time protocols that may be employed at the medium access control layer in order to support the new classes of services. We also describe a homogeneous approach to the implementation of medium access control protocols to support both connection-oriented and connectionless services, based on a uniform window splitting paradigm.

*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.

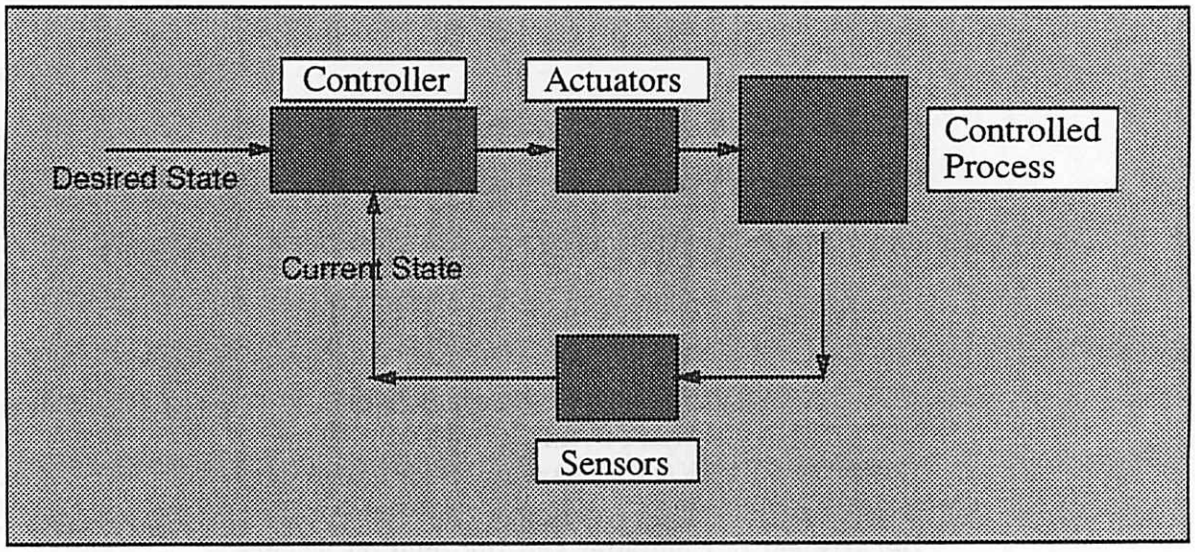

Figure 1 Model of a Real-Time Control System

1 *INTRODUCTION*

A real-time system is defined as a system whose correctness depends not only on the logical results of computation, but also on the time at which the results are produced (Stankovic and Ramamritham 1988). This is a general definition that encompasses a wide variety of systems including digital filters, multimedia communication systems, on-line transaction processing systems, message switching systems, manufacturing control systems and process control systems. In this paper, we restrict the scope of our attention to the last category of real-time systems. A real-time process control system (Figure 1), which constitutes the context for this paper, may be abstractly modeled as a feedback loop consisting of four components, viz., a controlled process, a controller, sensors and actuators. The sensors provide the controller with information about the current state of the controlled process. The controller is an information processing system that makes use of the information provided by the sensors to compute the actions required to reduce the disparity between the current state and the specified desired state of the system. The actuators realize the actions computed by the controller.

A simple illustrative example of a real-time control system is a servomechanism used to control the position of a motor shaft (the motor may in turn control the rotation of a robot arm joint). The controller in this system could be a microprocessor that performs a simple numerical computation such as determining the difference between the current position and the desired final position of the shaft and multiplying it by a constant. However in many real-time systems such as nuclear power plant control systems (Alger and Lala 1986), air traffic control systems (Cristian, Dancey and Dehn 1990), space mission control systems (Muratore et al. 1990, Strosnider 1988) and industrial process control systems (Martin 1967), the computations performed by the controller are more involved; the controller in the feedback control loop typically includes human components (in the form of one or more

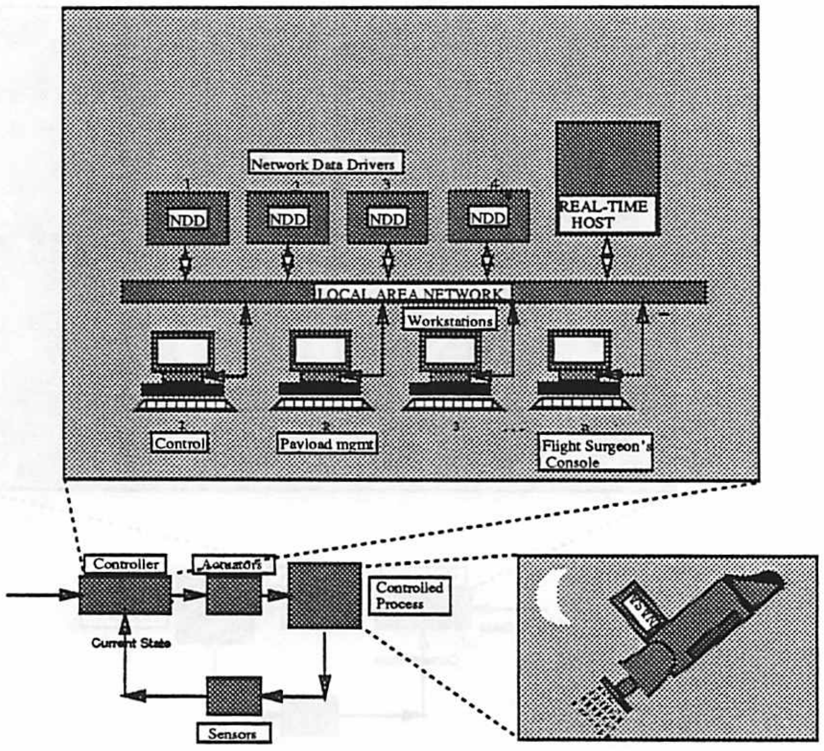

Figure 2 A Mission Control Scenario: A Distributed Real-Time System

human controllers), in addition to computers (for this reason, such systems are also known as *open loop* systems (Martin 1967)). These human controllers make decisions on the basis of raw or processed sensor data with the help of computers. The space shuttle mission control scenario depicted in Figure 2 (adapted from (Strosnider 1988)) is an example of such a system. In this system, network data drivers transmit raw telemetry data received from space to a real-time host computer for processing. Mission controllers responsible for various aspects of the mission, with the help of the host and the processed telemetry data, monitor and control the operation of the mission.

While systems such as the mission control system depicted in Figure 2 are typically large and distributed, they are not autonomous since many of the high-level activities are performed by cooperating human experts. A lot of work has already been done in the context of these open loop systems (Stankovic and Ramamritham 1988). The next logical step in the evolution of real-time systems is the introduction of autonomy. Real-time systems are steadily evolving towards the next generation of *closed loop* autonomous real-time systems (Iyengar and Kashyap 1989) in which human experts in the feedback loop are replaced by software. Figure 3 depicts an abstract model of such an autonomous control system. In this system, cooperating human experts are replaced by communicating problem-solving software tasks. In Japan (Whittaker and Kanade 1990), the Space Robot Forum, a prestigious group from Government, industry and academia, funded by the National Space Development Agency, recently outlined an ambitious schedule for "third-generation" space robotics, where machines work without much, if any, human intervention. This trend towards autonomy is driven both by techno-economic objectives such as enhanced productivity, prof-

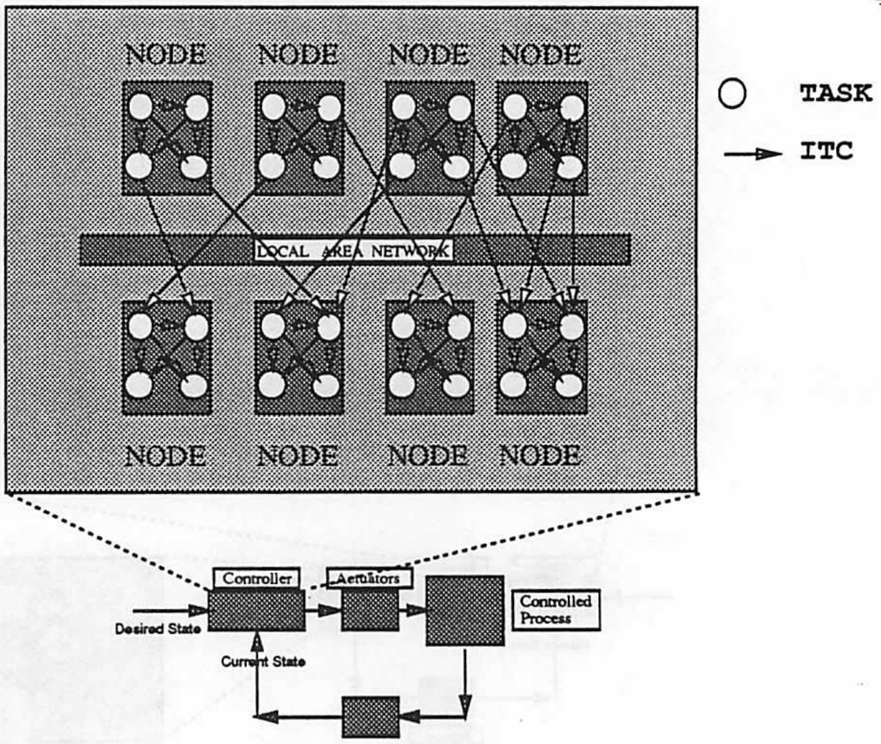

Figure 3 An Autonomous Real-Time Controller

itability and quality, and by a desire to relieve humans from dangerous, difficult and dull tasks (Iyengar and Kashyap 1989). The culmination of this trend towards autonomy would be the fully autonomous closed loop real-time control system. These real-time systems of the next generation will be distributed, complex, exhibit highly dynamic, adaptive and intelligent behavior and possess several types of timing constraints and operate in a highly non-deterministic environment (Stankovic 1988, Stankovic and Ramamritham 1990). Examples of harbingers of this trend towards autonomy are the NASA mission control system (Muratore et al. 1990), the Lockheed Pilot Associate system (Lark et al. 1990, Rouse, Geddes and Hammer 1990) and various ongoing projects in Robotics (Whittaker and Kanade 1990, Bares et al. 1989, Bihari, Walliser and Patterson 1989, Iyengar and Kashyap 1989, Weisbin et al. 1989).

In this paper, we address the communication requirements that arise in these complex closed loop real-time systems. The rest of the paper is organized as follows. In Section 2, we describe the distributed real-time system model that we assume as a basis in this paper. In Section 3, we discuss the approach to real-time communication in today's systems and illustrate the limitations of this approach and the requirements for future systems. In Section 4, we give a brief description of existing local area network architectures, and in Section 5, we propose RTLAN, a new local area network architecture for communication in distributed real-time systems. In Section 6, we describe the logical link control layer of this architecture. In Section 7, we describe the medium access control layer of RTLAN, and look at several MAC protocols that may be used at this layer to support the requirements of the LLC layer. In Section 8, we describe a uniform approach to implementing a homogeneous set of MAC protocols based on a window-splitting paradigm. In Section 9, we conclude the paper with a brief summary.

2 *SYSTEM MODEL*

The field of next generation real-time control systems is still evolving and we do not have sufficient experience to generate a precise set of requirements for such systems. We have adopted a general model that incorporates well accepted requirements for these systems. We describe our model below:

1. *Distributed System*:
   The system is distributed and is based on a local area network. The need for distribution will arise to satisfy performance, reliability and functional requirements. This trend towards distribution is evident even in today's real-time systems (Nielsen 1990, Muratore et al. 1990). Local area networks are typically used with distributed real-time systems because of the limited geographical span[1] of the system.

2. *Timing Constraints*:
   Activities in the system may have various types of timing constraints such as periodicity and deadlines associated with them.

3. *Dynamic Nature*:
   In addition to static periodic activities, the system will also be required to handle dynamically spawned activities.

4. *Predictability*: Predictability (Stankovic and Ramamritham 1990, Stankovic 1988) is considered an important requirement in real-time systems. For certain activities that are critical or essential, it is important to be able to determine whether the timing constraints of the activities will be met prior to actually accepting the activity for execution.

5. *Intertask Communication*:
   The functional requirements of the system will be realized through a set of tasks. Tasks will cooperate in order to achieve desired objectives. Cooperation will induce intertask communication requirements. Like all other activities in the system, intertask communication activities have timing constraints which will translate to individual message deadlines. We address this aspect in more detail in the next section.

[1]Even applications with a wider geographic span may be based on local area networks. For example, in air traffic control the entire air space is divided into smaller units (sectors) and the traffic within each unit is controlled by a distributed real-time system based on a local area network (Cristian, Dancey and Dehn 1990).

3 *REAL-TIME COMMUNICATION*

The term "real-time communication" may be used to describe any kind of communication activity in which the messages involved have timing constraints associated with them. For example, packet-switched voice communication, in which the individual voice packets have maximum delay constraints associated with them, is often termed real-time communication. However, in 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.

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

1. *Guarantee Seeking* Messages: These are messages typically critical or essential for the proper operation of the system. The requirements of these messages include a *guarantee* from the system that, if the activity that gives rise to them is accepted for execution, their timing constraints will be met with certainty.

2. *Best Effort* Messages: These are messages, typically with soft timing constraints, that do not require a *guarantee* from the system that their timing constraints will be met. However the system will try its best to satisfy the timing constraints of these messages, since minimizing the number of such messages whose timing constraints are violated will result in increased value (in some sense) for the system.

In the current generation of open loop distributed real-time systems (Strosnider 1988), the guarantee seeking messages are of two types, viz., *periodic* messages and *alarm* (alert) messages. Periodic messages, as the name implies, are messages that are transmitted periodically. They typically carry sensor information about the current state of the controlled process. For example, in an industrial process control system, computers at various sites in the plant *periodically* collect information about the state of the process such as flow rates, pressures and temperature, with the help of sensors, and transmit this state information to a central control room, where human controllers make decisions on the basis of this information. Periodic messages require guarantees that their delivery deadlines will be met in order to ensure that the actual state of the controlled process and the controller's view of the state obtained through these messages are close to each other. *Alarm* messages are used to disseminate information in an emergency situation. For example, in an emergency, the controller may have to shut down certain devices within a predetermined amount of time. Even though alert messages arise very rarely, due to their critical nature the system has to guarantee that alert messages will be delivered within their deadlines.

Examples of best effort messages in today's systems include some classes of commands from human controllers, and some classes of advisories and responses. For example, certain status and control messages, and trajectory advisory and response messages in the space shuttle mission control system (Strosnider 1988) may be classified as best effort messages. These messages occur asynchronously and are usually treated as soft real-time messages. The system is designed to minimize the response time for these messages.

Most current work in real-time communication (Lehoczky and Sha 1986, Le Lann 1987, Strosnider 1988, Strosnider and Marchok 1989) is based on the above model of commu-

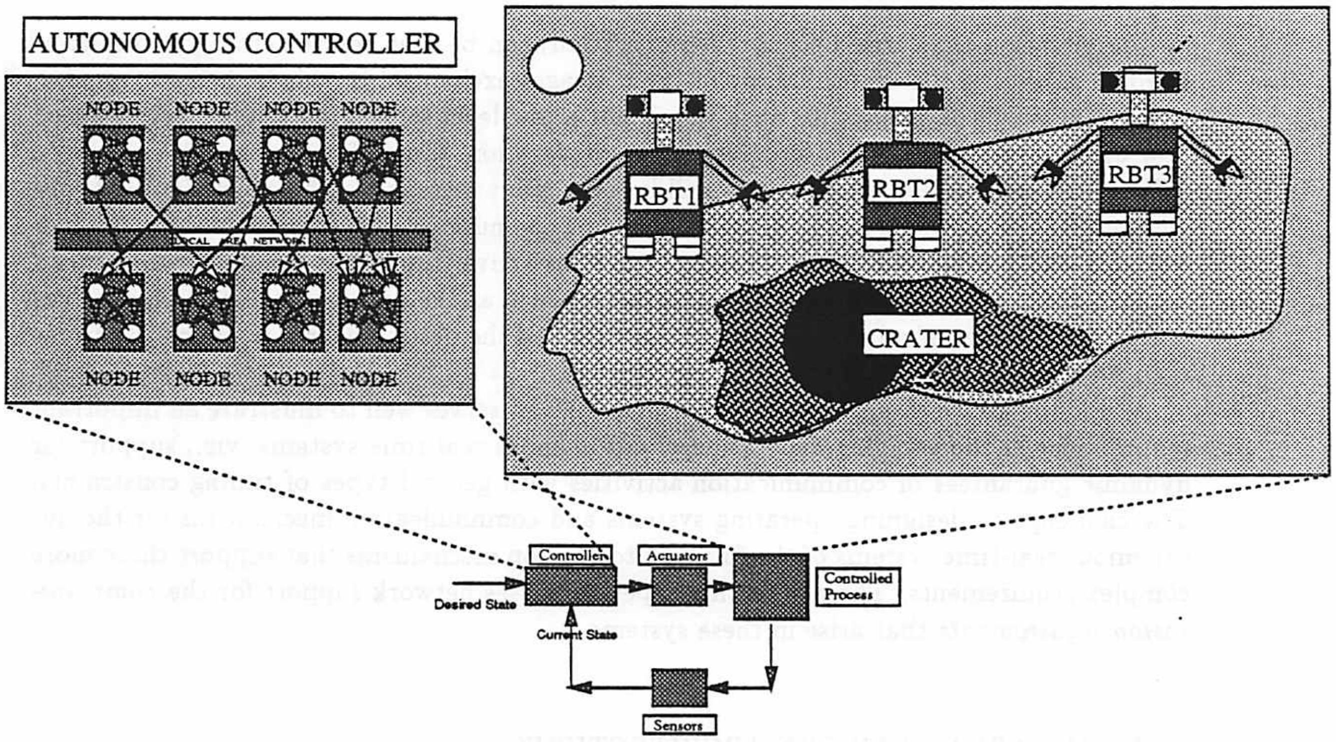

Figure 4 Cooperating Team of Robots

nication in distributed real-time systems, i.e., they assume that the guarantee seeking messages are either *periodic* or occur *rarely*. They also assume that the characteristics of these messages are *statically* specifiable. However this conventional model of communication requirements is likely to prove inadequate for the autonomous real-time systems of the future. These systems will be characterized by *dynamic* activities with *several* types of timing constraints (Stankovic and Ramamritham 1990). *Cooperation requirements* of distributed problem solving software that will replace cooperating human controllers, and distributed real-time operating system software (e.g., distributed scheduling (Ramamritham and Stankovic 1989, Ramamritham, Stankovic and Zhao 1989) will necessarily induce *richer communication patterns* than periodic state message communication. The system will have to provide mechanisms to predictably handle, in addition to the traditional communication requirements, *dynamically* spawned activities with communication requirements that will include *general* kinds of timing constraints. For example, each message in the set of messages involved in a distributed activity may have its own individual timing constraints that are independent of other messages. Consider the scenario depicted in Figure 4. A team of telerobots on a planet is coordinated by an autonomous mission controller. The object of the mission is to explore the planet. If the controller detects a crater near the current location of the robots that appears to be worth exploring, then it might decide to command the robots to explore the crater, if this (dynamically arising) activity can be accommodated into the system's schedule without violating the deadlines of other scheduled critical activities. In order to ensure predictability, the controller will first try to seek a *guarantee* that the timing requirements of this cooperative activity (including the timing requirements of all the messages that arise because of cooperation) can be met before actually committing itself to exploring the crater. Many of the messages exchanged in this high level cooperative activity will be aperiodic, since this is not a low level sensing activity. Depending on how essential they are to the progress of the exploration, some of the messages exchanged will require guarantees, while others will be best effort messages. The exact details of the communication involved in the activity, including the number of messages exchanged, their contents and individual timing constraints such as arrival times and deadlines will depend on both the activity and various dynamic factors such as the nature of the terrain near the crater, the number of robots assigned to this task and the desired degree to which the crater is to be explored.

While the above example may sound futuristic[2], it serves well to illustrate an important requirement of the evolving generation of autonomous real-time systems, viz., support for dynamic guarantees of communication activities with general types of timing constraints. The challenge in designing operating systems and communication mechanisms for the autonomous real-time systems of the future is to develop mechanisms that support these more complex requirements. The rest of this paper addresses network support for the *communication requirements* that arise in these systems.

4 *LOCAL AREA NETWORK ARCHITECTURES*

A computer network is a collection of geographically dispersed computers interconnected through a communication network. Depending on the geographic span of the network, a network may be classified[3] either as a local area network (LAN) or a wide area network (WAN). Distributed real-time systems are typically based on a LAN, and therefore we restrict our attention to LANs in this paper. A local area network is a network that spans a limited geographical area (0.1 km - 10 km) such as a building or a campus. A LAN is typically characterized by high speed, low error rates and ownership by a single organization.

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

Two different classes of services that can be offered by the various layers of a network are *connection-oriented service* (COS) and *connectionless service* (CLS) (Knightson, Knowles

---

[2]In fact, considerable effort is currently being invested to realize such autonomous systems (Bares et al. 1989, Iyengar and Kashyap 1989, Whittaker and Kanade 1990)

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

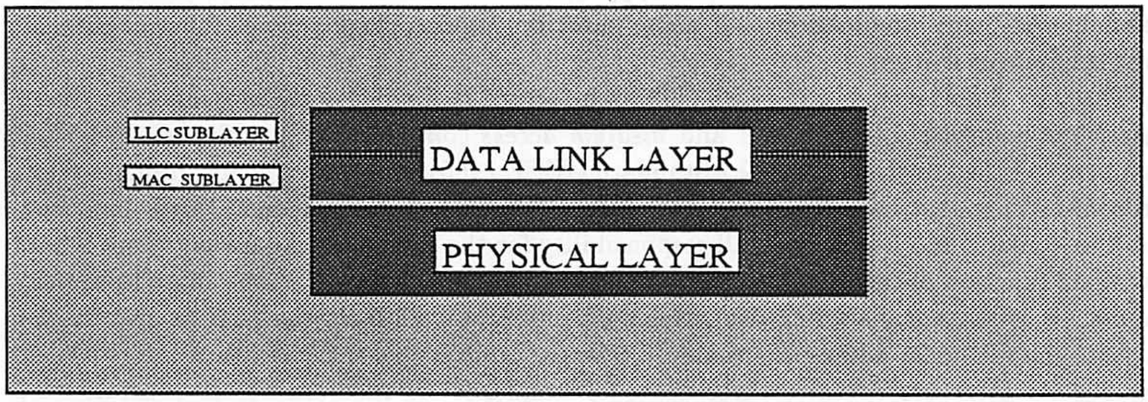

Figure 5 IEEE 802 LAN Architecture

and Larmouth 1988). COS is based on the establishment of a logical channel known as a connection. It is characterized by three phases, viz., *connection establishment*, *data transfer* and *connection release*. COS is typically suited for communication involving a long data transfer phase or a logically related sequence of messages, e.g., file transfer applications. Since a context is available (namely the logical connection) within which individual units of data passed between the communicating entities can be logically related, COS has smaller control overheads and can provide sequencing, flow control and error recovery. A network may use either a connectionless or connection-oriented mode of operation *internally* in order to provide communication services to its users. An *internal* connection in a wide area network that uses the connection-oriented mode of operation (e.g., TYMNET) internally is called a *virtual circuit*. For this reason, connection-oriented service is sometimes referred to as *virtual circuit service*.

The second category of communication services, connectionless service (CLS), as the name implies, is characterized by the absence of a logical connection between sender and receiver. There are no distinct phases since there is no connection to be established or released. Each unit of data is entirely self-contained and since there is no context in the form of a logical connection, the overhead information necessary to deliver the data to the receiver is duplicated in each unit of data. CLS is simpler and typically suited for short communications. However CLS does not provide sequencing, flow control or error recovery. In networks (e.g., ARPANET) that use the connectionless mode of operation *internally*, the independent packets involved in this operation are referred to as *datagrams*. Hence connectionless service is sometimes referred to as *datagram service*.

The architecture proposed by the IEEE Project 802 committee for local area networks (Figure 5) may be used to illustrate these concepts. This is a simple architecture that addresses only the lowest two layers in the OSI reference model, viz., the physical and data link layers. However the data link layer provides transport layer functionality in the case of an isolated (i.e., not internetworked) LAN and provides both connectionless (or Type 1) service and connection-oriented (or Type 2) service. These two classes of service are sufficient for many applications. If an application requires higher level functionality, it must implement it itself.

The data link layer in the IEEE 802 architecture is divided into two sublayers, viz., *logical link control* (LLC) sublayer and *medium access control* (MAC) sublayer. The *LLC sublayer*, specified in the IEEE 802.2 Standard (IEEE 1985), is responsible for implementing medium-independent data link functions, such as connection management, error handling and flow control, and has the overall responsibility for the exchange of data between nodes. The main function of the *MAC sublayer* is the management of access to the shared physical channel. It is responsible for transmitting data units received from the LLC layer over the physical channel after adding the required framing, addressing and checksum information. The 802 Architecture specifies three protocol standards for medium access control, viz., the CSMA/CD (802.3 standard), the token bus (802.4 standard) and the token ring (802.5 standard). The *physical layer* is responsible for the management of physical connections and for the transmission of bits over the transmission medium.

The services and protocols defined by the IEEE 802 architecture and other network architectures, although sufficient for many applications today, have an important limitation. Comer and Yavatkar (1989) point out that existing protocols do not make provisions for applications to specify their performance needs such as maximum delay, minimum throughput, maximum error rate etc., and existing network architectures do not have mechanisms to meet and guarantee these performance requirements. While they make this observation in the context of research in voice and video communication in future wide area networks, a similar observation may be made in the context of distributed real-time systems. It is important for tasks executing in a distributed real-time system to be able to specify their performance requirements including timing constraints of individual messages to the operating system, and for the operating system and the underlying network to provide support for meeting and guaranteeing these requirements. However existing operating systems and networks provide little support for this. For example, the Type 1 LLC service in the IEEE 802 architecture does not take timing constraints of messages into account explicitly, and the Type 2 service does not try to guarantee timing requirements of connections. Below, we propose the elements of RTLAN, a new local area network architecture for communication in distributed real-time systems, that alleviates this deficiency.

5 *RTLAN*

The RTLAN (real-time local area network) architecture is a local network architecture for communication in distributed real-time systems, that permits applications to dynamically specify their communication timing requirements and provides mechanisms to guarantee these requirements, if needed and if at all possible. RTLAN is targeted for complex embedded control applications and so we do not consider internetworking aspects. We therefore propose a simple four layer structure for RTLAN (Figure 6), along the lines of the IEEE 802 architecture. The four layers are the physical layer, the medium access control (MAC) layer, the logical link control (LLC) layer and the application layer. Some of the

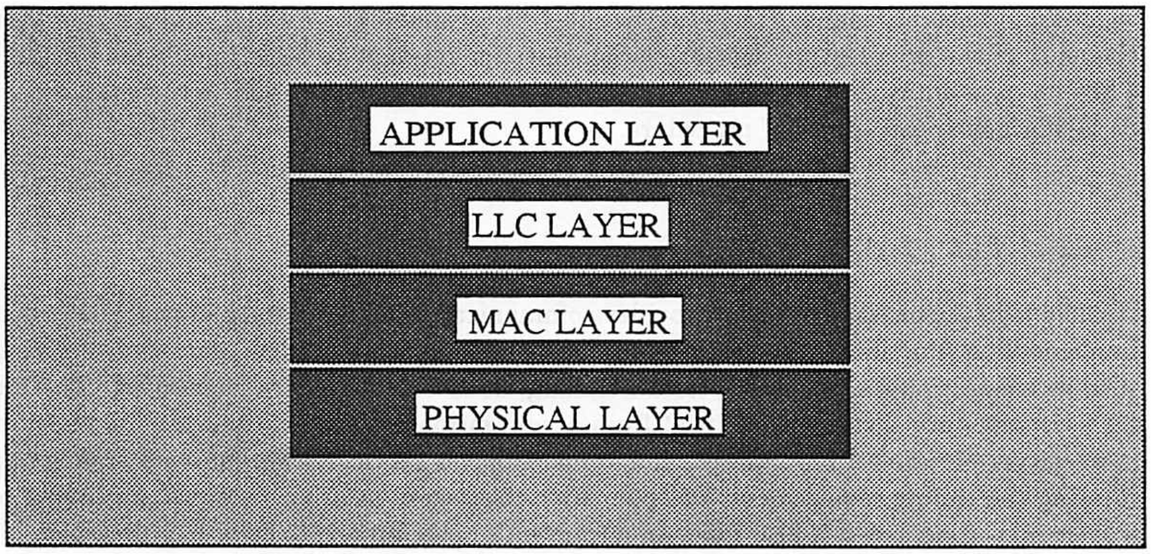

Figure 6 RTLAN Architecture

salient features of the RTLAN architecture are listed below:

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

2. *Time-Constrained Services*:
   RTLAN provides both connection-oriented and connectionless services, both of which consider the timing requirements of applications.

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

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

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

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

6 RTLAN LLC LAYER

The logical link control layer provides communication services to the layer above it by implementing functions that are responsible for medium-independent data link functions such as connection management, error handling, flow control and fragmentation. The RT-LAN architecture distinguishes itself from a conventional LAN architecture by providing new classes of connection-oriented and connectionless services, that provide support for meeting the timing requirements of application messages. In order to meet the timing requirements of messages, it also takes an unconventional approach to error handling and flow control. We describe these aspects of the LLC layer in the following sections.

6.1 SERVICES

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

6.1.1 REAL-TIME CONNECTION-ORIENTED SERVICE

RTCOS is a connection-oriented service that permits the sender to specify its timing requirements at the time of connection establishment. RTCOS is meant for supporting the requirements of the class of guarantee-seeking messages. The service is characterized by the establishment of a logical connection known as a *real-time connection*. A real-time connection (Figure 7) represents a simplex end-to-end communication channel between two communicating application level entities, a *sender* and a *receiver*. In order to set up a real-time connection, the sender specifies the timing requirements of the messages that it plans to send over the connection to the LLC layer at the time of connection establishment (connection establishment is done at the time of scheduling a task; the connection request is typically made by the operating system, which is also part of the application layer, on behalf of the sender). The timing constraints may be fairly general and may include periodicity, arrival time, laxity, deadline, etc. Figure 8 depicts an example of the requirements that may be specified by an application task to the LLC layer. In this example, the application task requires guarantees that the timing constraints of a session involving four messages will be satisfied. The first message is periodic and a guarantee is requested for N1 consecutive instances of the message. The remaining three messages are aperiodic with various arrival time and deadline requirements. As in a typical connection-oriented service, the service provider tries to set up the requested connection through a process of negotiation that may

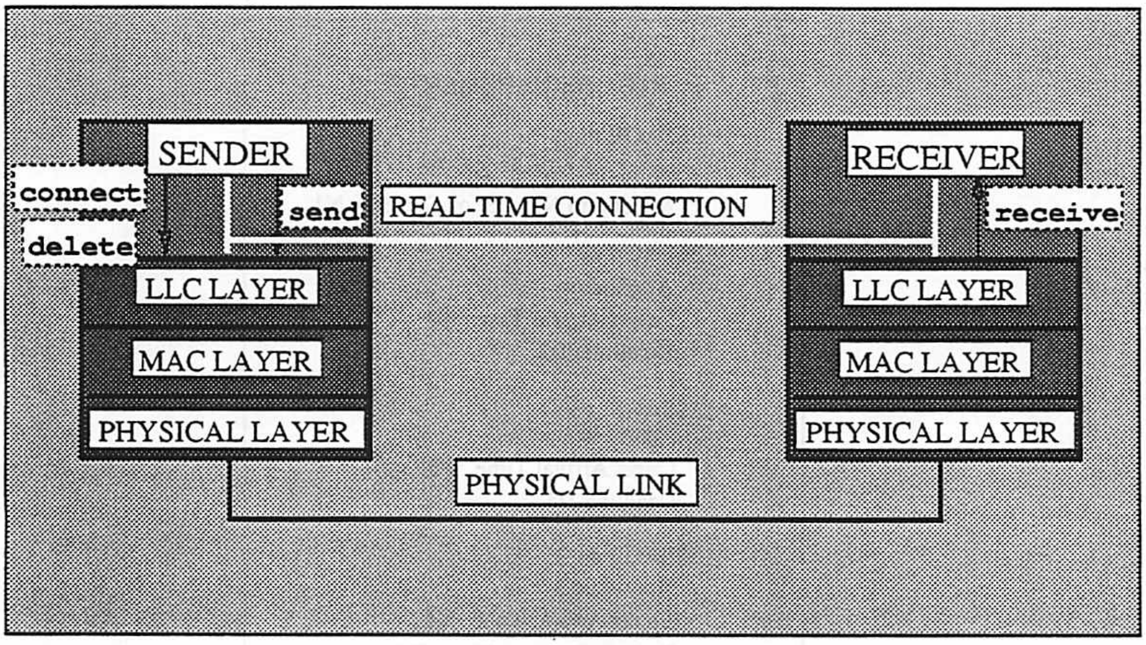

Figure 7 Real-Time Connection

involve the sender and the receiver in addition to itself (Knightson, Knowles and Larmouth 1988). The connection is set up only if the specified requirements can be guaranteed; otherwise the sender is informed that the connection cannot be established. In order to set up the connection, the RTCOS service provider employs the services provided by the MAC layer and suitable scheduling algorithms, which we discuss later. The following operations that comprise real-time connection-oriented service summarize the above descriptions:

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

- **send(rtcid, message)**
  Sender requests delivery of a message to the receiving end of the specified real-time connection.

- **message ← receive(rtcid)**
  Receiver requests receipt of a message sent over the specified real-time connection.

- **delete(rtcid)**
  Sender or receiver requests termination of specified real-time connection.

6.1.2 REAL-TIME CONNECTIONLESS SERVICE

RTCLS is an unreliable *connectionless* service used for transmitting time-constrained messages. It is unreliable in the sense that the timing constraints of messages transmitted using this service may not be satisfied. However RTCLS tries to deliver messages within their timing constraints on a *best effort* basis. Thus this service is suitable for the class of

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 8 Real-Time Connection Requirements

best effort messages. By best effort, we mean that at each decision making point within the service, decisions are made on the basis of timing constraints of the pending packets. For example, if there are several packets waiting to be transmitted, then the system would try to transmit them in an order that minimizes the number of messages whose deadlines are not met. Since real-time connectionless service does not involve setting up a connection, it is defined by a simpler set of operations:

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

- **(message, sender) ← receive()**
  Receiver requests receipt of a message.

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

6.2 FRAGMENTATION

One of the functions implemented by the logical link control layer is the transformation of messages provided to it by the application layer to a form suitable for the medium access control layer. One step involved in this function is known as *fragmentation* or *packetization*.

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

6.3 GUARANTEEING REAL-TIME CONNECTIONS

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

6.3.1 PRIORITY ASSIGNMENT APPROACH

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

This approach is based on the ***rate monotonic*** priority assignment scheme (Liu and Layland 1973). 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 (1973) 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

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

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 (1986) 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***

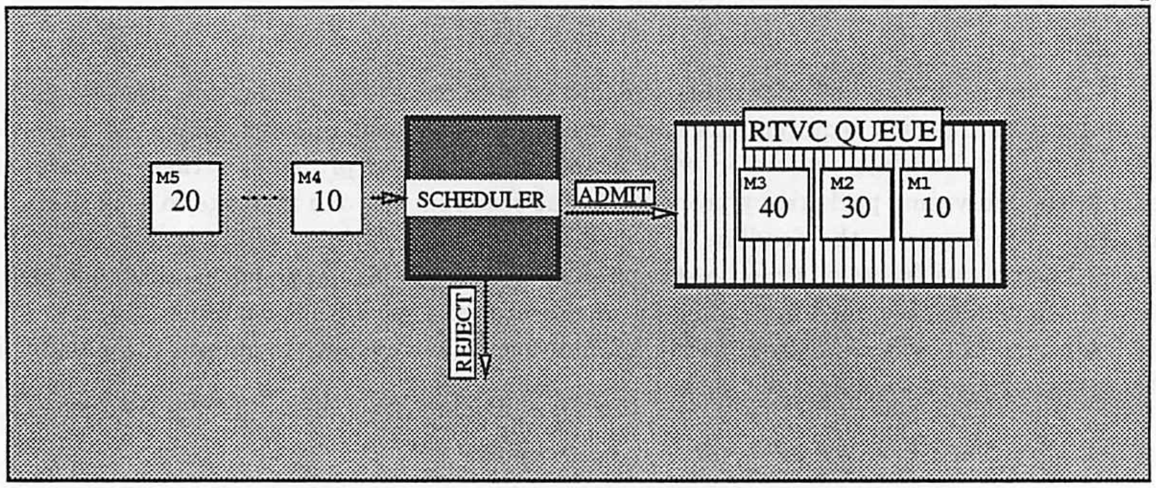

Figure 9 Real-Time Virtual Circuit Scheduling

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.

6.3.2 REAL-TIME VIRTUAL CIRCUIT APPROACH

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

The LLC layer makes use of RTVCs as follows. Each RTVC has a transmission queue associated with it. When an application entity requests a real-time connection 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* (Cheng, Stankovic and Ramamritham 1988) to determine if the set of fragments can be inserted (according to some insertion

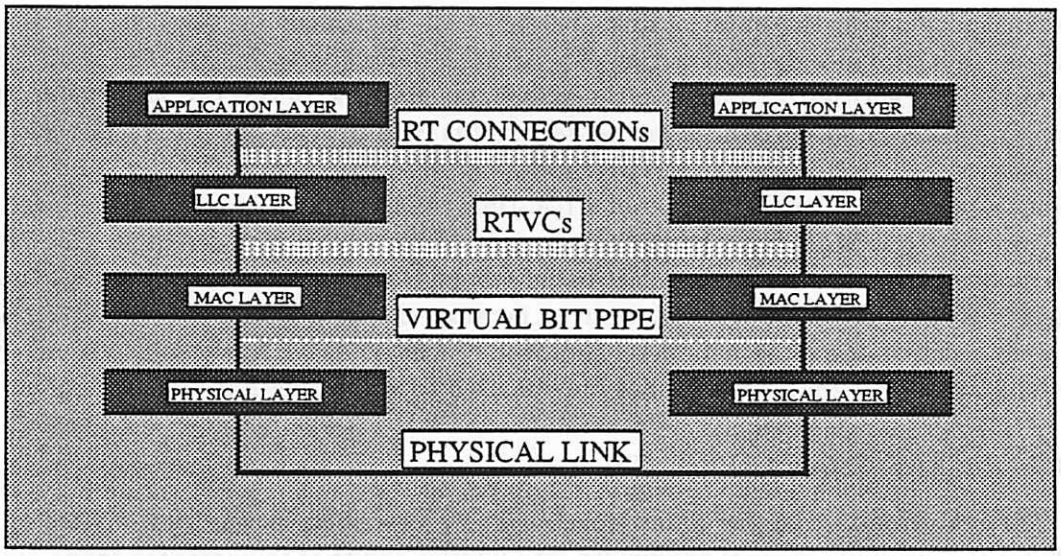

Figure 10 Abstractions provided by RTLAN layers

discipline, such as the first-in first-out (FIFO) or the minimum laxity first (MLF) discipline[4]) into the queue associated with some RTVC, without violating the timing constraints of the packets that have already been admitted into the queue and that are awaiting transmission. In order to make this determination the scheduler makes use of a worst case assumption, since the scheduler cannot predict the service time exactly. The assumption made is that each packet in the queue will have a service time equal to the worst case service time. Such a worst case service time is guaranteed to exist, since an RTVC by definition has a bounded packet service time. The example shown in Figure 9 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 transmitted. Packet M1 has a laxity (time until deadline for start of transmission) of 10, M2 has a laxity of 30 and M3 has a laxity of 40. If a packet M4 of laxity 10 arrives, then it cannot be admitted into the system, if the MLF discipline is used, since the deadlines of both M1 and M4 cannot be simultaneously met. However a packet M5 with a laxity of 20 can be admitted, since it is possible to meet its laxity requirements without violating the requirements of any of the messages already in the queue. It should be pointed out that, for the real-time virtual circuit approach to guarantee a reasonable fraction of the connection requests that are made, the worst case packet service time must be of the same magnitude or smaller than the average laxity of the packets involved.

Note that RTVCs are abstractions provided by the MAC layer to the LLC layer, Fig-

[4]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.*

ure 10 illustrates how each layer in the architecture provides an abstraction that is used by the layer at the immediately higher level to implement its services. The physical layer receives a bit stream from the MAC layer and converts the bits into *electrical signals* that are transmitted over the physical link. The physical layer hides the physical details of the link and thus provides the abstraction of a virtual *bit pipe* (Bertsekas and Gallager 1987) to the MAC layer entities. Protocols in the MAC layer make use of this raw *multiple access* bit pipe with *potentially unbounded packet service times* to provide the abstraction of logical channels with bounded packet service times, viz., RTVCs. The LLC layer employs these channels to provide the abstraction of a real-time connection to the application layer.

Note that the MAC layer has to employ suitable protocols in order to provide such an abstraction. For instance, an Ethernet based system is unsuitable, since a message that has entered service may *never* get transmitted because of the randomized collision *resolution* strategy used in the Ethernet MAC protocol. In Section 7.1.2, we will look at several medium access control protocols that may be used to realize the RTVC abstraction.

6.3.3 RESERVATION APPROACH

The real-time virtual circuit approach that we described in the previous section distributes the available channel bandwidth into RTVCs and tries to guarantee connections on individual RTVCs based only on local knowledge of connection requests. This forces each node to *pessimistically* assume a *worst case* packet service time for each packet. This can result in connection requests that can actually be guaranteed to be unnecessarily rejected. The reservation approach tries to make use of *global knowledge* of all the connection requests in the system, thereby increasing the chances of guaranteeing connections.

The reservation approach treats the entire channel as a single schedulable entity on which the LLC layers at the various nodes try to schedule their connection requests. Such an approach requires that the LLC layers have a *system-wide* view of all the accepted reservation requests made at *any* node in the system. With such a global view, the LLC layer at a node can apply a scheduling algorithm based on an appropriate insertion discipline (as in Section 7.3.2) to determine whether a connection request can be accepted or not. However, since the scheduling algorithm has global knowledge of all the packets (in all the nodes) in the system that are awaiting transmission, it does not have to make worst case assumptions about the service time of a packet. Thus the chances of guaranteeing connections can be expected to be better under this approach.

Implementation of such a system-wide view requires special support from the MAC layer in the form of suitable MAC protocols. We discuss approaches to implementing such a queue in Section 7.1.3.

6.4 FLOW CONTROL AND ERROR CONTROL

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

In the real-time connection oriented service, flow control schemes based on sliding window protocols where the receiver may block the sender by withholding acknowledgements, are inappropriate. If a receiver can block the sender for an arbitrary amount of time (because of an insufficient number of buffers), then the timing constraints of guaranteed packets may be violated. In order to guarantee the timing constraints of packets, the required number of buffers will have to be *reserved* at the destination node. This will permit the sender to transmit its packets whenever they are ready, rather than wait for permission from the receiver. This *buffer reservation* is part of the three way negotiation involved in setting up a real-time connection. When a connection is requested under any of the approaches described in Section 6.3, the LLC layer makes sure that sufficient buffer space is reserved at the destination nodes for the packets involved in the request. In a real-time connectionless service, flow control can be exercised by merely dropping packets that arrive when the receiver's buffers are full. This is acceptable, since RTCLS does not offer any guarantee on the delivery of packets.

Two kinds of error control schemes may be used for RTCOS. The first scheme is an ARQ scheme based on *temporal redundancy* that is appropriate when the deadlines of the packets involved in a connection are large enough to permit the use of timeouts. At the time of connection establishment, the maximum number of retransmissions possible for each packet is computed based on the packet's deadline and the retransmission timeout (alternatively, this number may be included in the specification of the requirements for the connection). The scheduler takes into account all the potential retransmissions in determining whether the requested connection can be guaranteed or not. When the packet is actually transmitted, the receiver returns an acknowledgement (transmitted as a real-time datagram) to the sender. If the acknowledgement is received within the retransmission timeout period, then all the remaining duplicates are dequeued; otherwise, the next duplicate is transmitted, when its turn comes. A variation on this scheme is to use temporal redundancy and avoid acknowledgements all together (Kopetz 1983). In this variation all the duplicates are transmitted, irrespective of whether or not the earlier duplicates suffer transmission errors, leaving it to the receiver to select an error-free duplicate and discard the others.

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

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

7 RTLAN MAC LAYER

A local area network is typically based on a shared physical channel such as a bus or a ring, commonly referred to as a multiple access channel. Since multiple nodes may simultaneously contend for access to this shared channel, a mechanism is needed to manage access to it. The main function of the medium access control layer is to arbitrate access to the channel. The MAC layer implements this arbitration mechanism through a suitable *multiple access protocol*. Kurose, Schwartz and Yemini (1984) provide a good survey of multiple access protocols. Multiple access protocols can be classified into *real-time* and *non-real-time* protocols on the basis of whether or not they provide support for real-time communication. Many protocols that have been proposed in the literature, such as the Aloha protocol (Abramson 1970) the CSMA protocols in (Kleinrock and Tobagi 1975) and the Ethernet protocol (Metcalfe and Boggs 1976) are non-real-time protocols since they do not incorporate any notion of packet timing constraints. In this section, we look at examples of real-time protocols that have been proposed in the literature that may be used to provide support for RTCOS and RTCLS.

7.1 RTCOS/MAC PROTOCOLS

In Section 6.1.1, we defined RTCOS as a connection-oriented service that permits an application layer entity to specify its communication timing requirements to the LLC layer and seek a guarantee that these requirements will be satisfied. We also described three different approaches to providing such guarantees, viz., *priority assignment* approach, *real-time virtual circuit* approach and *reservation* approach. The LLC layer invokes the services of the MAC layer in order to provide such guarantees. In this section, we look at several MAC protocols that may be used to support the services required by the LLC layer, for each of these approaches. The priority assignment approach requires priority resolution protocols at the MAC layer; the real-time virtual circuit approach requires MAC protocols that can guarantee bounded packet service times and the reservation approach requires reservation protocols.

7.1.1 PRIORITY RESOLUTION PROTOCOLS

In Section 6.3.1, we described a priority-based scheduling approach based on the rate monotonic algorithm and the extensions to this algorithm proposed by Lehoczky and Sha (1986), and Strosnider (1988). This approach may be used to guarantee certain classes of statically specifiable messages. In order to ensure that packets are actually transmitted according to the priorities assigned to them by the LLC layers, the MAC layers will have to employ an appropriate *priority resolution protocol*. A priority resolution MAC protocol always selects for transmission, the packet with the highest priority among all the contending packets in the system.

The IEEE 802.5 standard for token ring local area networks includes a priority resolution scheme. In a token-passing ring network, access to the ring is arbitrated through a short control packet known as a token. At any moment, only the node that is in possession of the token is permitted to transmit. A node that has completed transmission or that has no packets to transmit passes on the token to the next node. This simple token-passing scheme may be augmented to support priority resolution by including additional fields in the token. In the IEEE 802.5 standard, priority resolution is supported using two 3 bit fields in the token known as the *priority* field and the *reservation* field. The priority resolution scheme works as follows. When a station captures the token, it sets a one bit field in the token known as the *token bit* to indicate that the token has been *claimed*. It then transmits the claimed token followed by the data packet. Each node examines the claimed token as it passes by. It overwrites the reservation field in the token with the priority of its pending packet, if this priority is greater than the value in the reservation field. Thus the reservation field contains the priority of the packet with the highest priority among all the packets at the heads of MAC transmission queues at the various nodes in the system. After the transmitting station has received the token and the data packet back, it removes the data packet from the ring and clears the token bit in the token. It also copies the reservation field of the token to the priority field before releasing the free token. Any node with a pending packet with priority greater than or equal to the priority field of the free token may now capture the token. Strosnider and Marchok discuss the use of the token ring priority resolution scheme for scheduling statically specified message sets in more detail in (Strosnider and Marchok, 1989).

Priority resolution protocols for bus-based systems are classified by Valadier and Powell (1984) into three categories, on the basis of the underlying scheme involved:

1. *Deference delays*: In this scheme at the end of a packet transmission, each node defers transmission of its packet (if any) by a period of time whose length in round trip propagation delay units is equal to $p_{max} - p$. Here p is the priority associated with a node and p_{max} is the largest priority value possible. At the end of its deference period, each node senses the channel. If the channel is sensed to be idle, it means that no other node has a value of p greater than that of the node that sensed the channel. So this node is allowed to transmit its packet. Thus this scheme selects the node with the largest value of p (highest priority) that has a packet to transmit. Note that the length of the deference delay, the worst case overhead (in the form of wasted channel time) per packet that arises because of priority resolution, lies in the range $[0, p_{max} - p_{min})$. The average overhead, assuming all possible values of the overhead are equally likely, is given by $\frac{1}{2}(p_{max} - p_{min})$, i.e., it is a linearly increasing function of the number of priority levels. The protocols proposed by Franta and Bilodeau (1980) and Tobagi (1982) are examples of this class.

2. *Preamble lengths*: In this scheme, at the end of a packet transmission, each node transmits a preamble signal for a period of time whose length in round trip propagation delay units is equal to $p - p_{min}$, where p_{min} is the smallest priority value possible.

Thus a node with a higher priority will transmit a longer preamble signal. Collision detection is suppressed at a node during the period that its preamble is being transmitted. If a node detects a collision at the end of the transmission of its preamble, it means that there is another node with a larger value of p that is still transmitting its preamble. Only the node which does not detect a collision at the end of its preamble, i.e., the node with the longest preamble length or the largest value of p, is allowed to transmit its packet. The per packet overhead that arises because of priority resolution again lies in the range $[0, p_{max} - p_{min}]$ units of time. Thus the average overhead is again a linearly increasing function of the number of priority levels. The protocol proposed by Iida et al. (1980) is an example of a protocol that employs this scheme.

3. *Forcing Headers*: The forcing header scheme makes use of the inherent wired-OR property of a broadcast bus, namely the property that the value of the bit received by a node from the bus is equal to the logical OR of the bits transmitted by all the nodes. In this scheme, each packet transmission is preceded by a header epoch, during which all the nodes with packets to transmit, simultaneously transmit their priorities from the most significant to the least significant bit in successive slots (of length equal to a round trip propagation delay). The nodes sense the medium as they transmit their priority bits. If a node has transmitted a 0 bit in a slot and receives a 1 bit, it means that some node with a higher priority value is contending for access to the channel. In this case the node that transmitted the 0 bit drops out of contention. At the end of the header epoch, only the node with the highest priority value remains and this node transmits its packet to completion. The length of the header epoch is equal to the number of bits required to represent the range of values $[p_{min}, p_{max}]$, i.e., $\lceil \log_2 (p_{max} - p_{min} + 1) \rceil$ time units. This represents the priority resolution overhead and is constant for all packets and hence also represents the average overhead. Thus the average overhead for this scheme increases ***logarithmically*** with the number of priority levels. The MLMA or multilevel multiaccess protocol (Rothauser and Wild, 1977) employs such a scheme.

It is clear that, among the above schemes for bus-based systems, the scheme based on forcing headers incurs the least overhead, because of the logarithmic relationship between the overhead and the number of priority levels in this scheme. In fact, such a scheme is used in the Distributed Systems Data Bus (DSDB) developed by IBM, which forms the hub of a large distributed real-time system (Strosnider 1988). In Section 8.1, we describe a priority resolution protocol based on a window splitting paradigm whose overheads are about the same as the forcing headers scheme.

7.1.2 REAL-TIME VIRTUAL CIRCUIT PROTOCOLS

A real-time virtual circuit is a logical channel that has the property that the service time of a packet queued on the channel is bounded. The packet service time consists of two components, the physical *channel access time* and the packet *transmission time* (Figure 11). The channel access time arises because the underlying physical channel in a local area network is typically a multiple access channel that is shared by all the nodes on the network; a node may therefore have to wait for the channel arbitration mechanism to permit it to

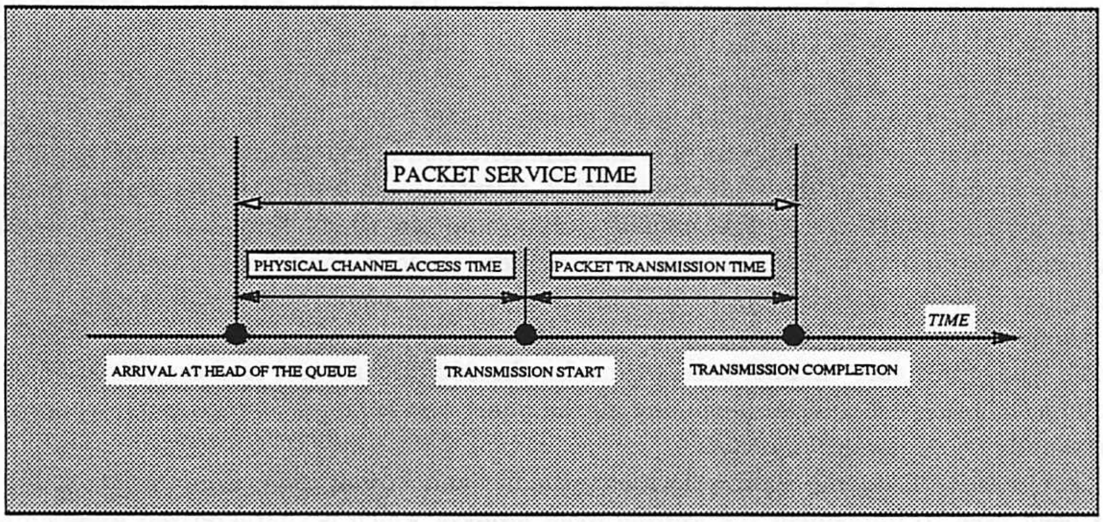

Figure 11 Packet Service Time

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

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

In token-passing protocols, the nodes are organized in a logical (e.g., token bus (IEEE 1982)) or physical ring (e.g., token ring (IEEE 1986)). A real (e.g., token ring, token bus) or virtual (e.g., BRAM (Chlamtac, Franta and Levin 1979), MSAP (Kleinrock and Scholl 1980) token that circulates around this ring is used to arbitrate access to the channel. The token confers upon its holder the privilege to transmit on the channel for a bounded amount of time (the token holding time). If there are no packets to transmit, or if the token holding time has been exceeded, the token is passed onto the next node in the ring. If the token holding time per node is P time units (i.e., a node is permitted to transmit only one packet in each cycle), then the maximum length of the interval between successive channel accesses by a node is given by $N(P + \tau)$, where N is the number of nodes in the system and τ is the token passing overhead.

The 802.3D protocol proposed by Le Lann (1987) is another MAC protocol that may be used to guarantee a bounded channel access time. The '802.3' in the name refers to the IEEE 802.3 MAC layer standard (IEEE 1985) for CSMA/CD (the access control method used in Ethernet) which, owing to the collision resolution strategy (binary exponential backoff) used, cannot guarantee an upper bound on the channel access time. The 'D' refers to the fact that the 802.3D protocol is deterministic, i.e., it can guarantee an upper bound on the channel access time. This protocol is essentially an adaptive tree walk protocol. Normally, the nodes are in the *random access* mode in which they access the channel randomly without any coordination. Thus under low load conditions, channel access is immediate. However, the moment a collision occurs, the nodes switch to the *epoch* mode in which a tree search process is used to resolve the collision within a bounded amount of time. The worst case channel access time for the 802.3D protocol is the worst case time required to resolve a collision involving all the nodes and is equal to $N(P + \tau)$, where τ is the round trip propagation delay.

The notion of waiting room priorities (Valadier and Powell 1984, Ramamritham 1987) 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 (Valadier and Powell 1984).

Thus a variety of paradigms may be used to realize the abstraction of a real-time virtual circuit, all with approximately the same worst case channel access time (except the waiting room protocol that has a larger worst case channel access time). In Section 8.2, we describe another protocol, this time based on a window splitting paradigm, that has approximately the same bound on the channel access time as most of the protocols above, but which also provides structural homogeneity.

7.1.3 RESERVATION PROTOCOLS

In Section 6.3.3, we described the reservation approach to implementing RTCOS, in which LLC layers make use of global knowledge of all the connection requests in the system in deciding whether a connection request can be accepted or not. We also pointed out that such an approach requires that the LLC layers have a global view of all the accepted reservation requests made at *any* node in the system. In a distributed system, such a global view may be realized through a replicated, globally consistent queue of reservation requests.

One approach to maintaining such a global queue is to broadcast each arriving connection request to all the nodes, during specific time slots (reservation slots) dedicated for this purpose. Schedulers at the LLC layer of each node execute identical scheduling algorithms to determine if a connection request is to be accepted or rejected, and update the local copy of the global queue accordingly. In a broadcast channel, in the absence of transmission errors or component failures, since all the nodes receive the same request messages at the

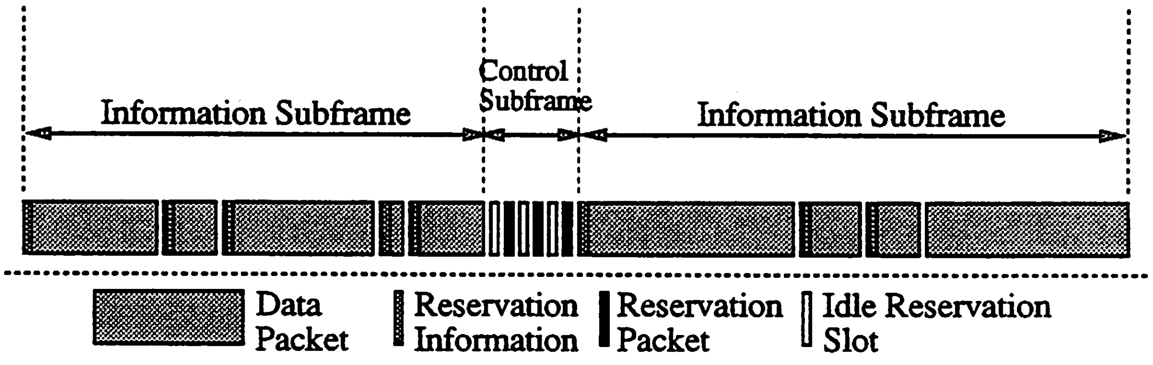

Figure 12 Priority-Oriented Demand Assignment

same time, and all the schedulers apply the same scheduling algorithm, the local copies of the queue at each node remain consistent.

The Priority Oriented Demand Assignment (PODA) reservation protocol proposed by Jacobs et al (1978) in the context of satellite-based communication is an example of a MAC protocol that uses this approach. In this protocol, channel time is divided into frames each of which consists of an *information subframe* and a *control subframe* (Figure 12). The information subframe is used for scheduled packet transmissions. These transmissions may contain additional reservation requests in their headers. The control subframe is used to broadcast reservations that cannot be sent as part of the data transmissions in a timely manner (e.g., when a station wants to make a reservation for a high priority message or when a station wants to join the system). In the PODA protocol, all stations maintain a local copy of a scheduling queue ordered by reservation urgency, which is a function of the delay class and priority of a message. Reservations with the same urgency are ordered further to ensure fairness to all the stations involved. Channel time in the information subframe is allocated to the stations according to this queue. Whenever a reservation request for a message is broadcast, all the nodes that successfully receive the message enter the request in their respective local scheduling queues (on the basis of its reservation urgency).

The main problem with a reservation protocol such as PODA is the lack of robustness. In the presence of transmission errors and failures, the local copies of the global queue at each node may become mutually inconsistent resulting in possible simultaneous channel access by more than one node. This would cause collisions and a violation of the guarantees offered. In order to ensure proper operation despite failures and errors, additional mechanisms for fault-tolerance are required.

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

An alternative approach is to employ a synchronous atomic broadcast protocol (Cristian 1990), again based on redundancy. Such a protocol guarantees the following three properties:

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

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

The reservation approach is a good way of avoiding the pessimism inherent in the real-time virtual circuit approach that makes use of worst case assumptions. However, as we saw above, it incurs high implementation complexity in the form of special mechanisms for fault-tolerance.

7.2 RTCLS/MAC PROTOCOLS

In Section 6.1.2, we defined RTCLS as an unreliable connectionless service that tries to deliver messages within their timing constraints on a best effort basis. In order to provide such a service to the application layer, the LLC layer invokes suitable MAC layer services that are implemented through a class of MAC protocols which we refer to as *best effort* protocols. Best effort protocols take into account the timing constraints of the individual contending packets in arbitrating access to the shared channel. Even though these protocols do not provide the ability to guarantee that the timing constraints of messages will be satisfied, they try to minimize the number of messages that are *lost*, i.e., that do not meet their deadlines.

The window protocol proposed by Kurose, Schwartz and Yemini (1983) is an example of a protocol that considers timing constraints of messages in arbitrating channel access. The protocol assumes that all packets have identical laxities, as would be the case in a voice communication application. The protocol effectively implements a global first-in first-out (FIFO) policy of message transmission augmented with an additional policy element that dictates the discarding of messages that have missed their deadlines before transmission.

The FIFO policy is appropriate when all the packets have identical laxities. This policy chooses the packet that arrived first, i.e., the packet that has waited longest and hence is likely to miss its deadline first (the most "urgent" packet). However other policies have also been proposed in the literature. For example, Kallmes, Towsley and Cassandras (1989) have considered the LIFO (Last In First Out) policy, and shown that under certain conditions (when the deadlines are i.i.d. with concave CDFs) this policy is optimal. In (Panwar, Towsley and Wolf 1988) 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 (Zhao and Ramamritham 1987) and the MLF window protocol (Zhao, Stankovic and Ramamritham 1990) 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 (Zhao and Ramamritham 1987), each node maintains two clocks, a real time clock and a virtual time clock that runs at a higher rate than the real-time clock. Whenever a node senses the channel to be idle, it resets and restarts its virtual-time clock. When the reading on the virtual clock is equal to some parameter value of a message waiting to be transmitted, the node transmits the message. Collisions are resolved through a random backoff procedure. Different transmission policies are implemented by using different message parameters to control the operation of the virtual clock. For example, choice of message arrival time as the parameter used by the protocol corresponds to the first-in first-out transmission policy (Molle and Kleinrock 1985), while use of message laxity corresponds to the minimum laxity first policy.

The MLF window protocol belongs to the class of inference-seeking protocols (Zhao, Stankovic and Ramamritham 1990). In the window protocol, a window that slides along the real-time axis is used to identify the node that has the message with the most urgent transmission requirements (the message whose 'latest time to send' is smallest) and this node is granted transmission rights on the channel. The protocol effectively tries to implement a global minimum laxity first policy. In this protocol, each node maintains a window that spans a segment of the real-time axis. When a node has a message to transmit, it waits for the channel to become idle and then transmits the message, if the latest time to send the message falls within the current window. If there is a collision (which can occur if another node also had a message that fell within the window), then all the nodes split the window into two halves and examine the left window first. If there is only one node with a message in the left window, then that node transmits its message successfully. If there are two or more nodes in the window, then a collision occurs again and the splitting process is repeated until a message is successfully transmitted. If there are no nodes with a message whose latest time to send falls within the left window, the window is expanded to include the left half of the right window and the window examination process is repeated again. This process is continued until a message is transmitted, or the whole of the initial window has been examined. The window protocol has been shown to very closely approximate an ideal protocol that implements, without incurring any arbitration overheads, the minimum laxity first policy for message transmission (Zhao, Stankovic and Ramamritham 1990). An important advantage of this protocol over traditional window protocols is that a newly arriving message need not wait for all the messages currently involved in contention resolution to be transmitted, before being considered for transmission.

8 *A UNIFORM APPROACH TO MAC PROTOCOLS*

In the previous section, we looked at several kinds of MAC protocols that may be used to support RTCLS and RTCOS, including forcing header protocols, token-passing protocols, tree walk protocols, virtual time protocols and window protocols. Each of these protocols is based on a particular distinct medium access control paradigm and is suitable for a particular kind of communication service. In (Arvind, Ramamritham and Stankovic 1990), we have proposed homogeneous MAC protocols, all based on a uniform window-splitting medium access control paradigm, to support RTCOS and RTCLS.

The starting point for this work was provided by the MLF window protocol proposed by Zhao, Stankovic and Ramamritham (1990). 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. In (Arvind, Ramamritham and Stankovic 1990) we have generalized and extended this protocol and developed uniform window protocols to support both RTCLS and RTCOS. The unifying thread that is common to all the new window protocols is a contention resolution 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 $p \in [p_{min}, p_{sup})$ that can 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. PBCR may be implemented using a window splitting paradigm as follows. Each node maintains a data structure known as a window that is characterized by the current position π of its left edge and its current size δ. The window at any moment spans the range of values $[\pi, \pi + \delta)$. If the parameter p associated with a node lies in the range of values spanned by the window, the node is said to be in the window. At the end of a packet transmission epoch, each node that lies within the initial window $[p_{min}, p_{min} + \delta_{max})$, where δ_{max} is the maximum size the window is allowed to assume, starts transmitting its packet. If there is only one node in the window, then that node continues to transmit its packet to completion; if two or more nodes lie within the window, then the window is split into two and the protocol is recursively repeated with the left half of the window, and if necessary again with the right half of the window, until a packet is transmitted successfully. In the example shown in Figure 13, 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 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 window is therefore split into two and the left half considered first. Only one node, the node with the smallest value of p, lies within this half and it transmits to completion. This window splitting paradigm may be used to implement MAC protocols that can support RTCOS and RTCLS as described below.

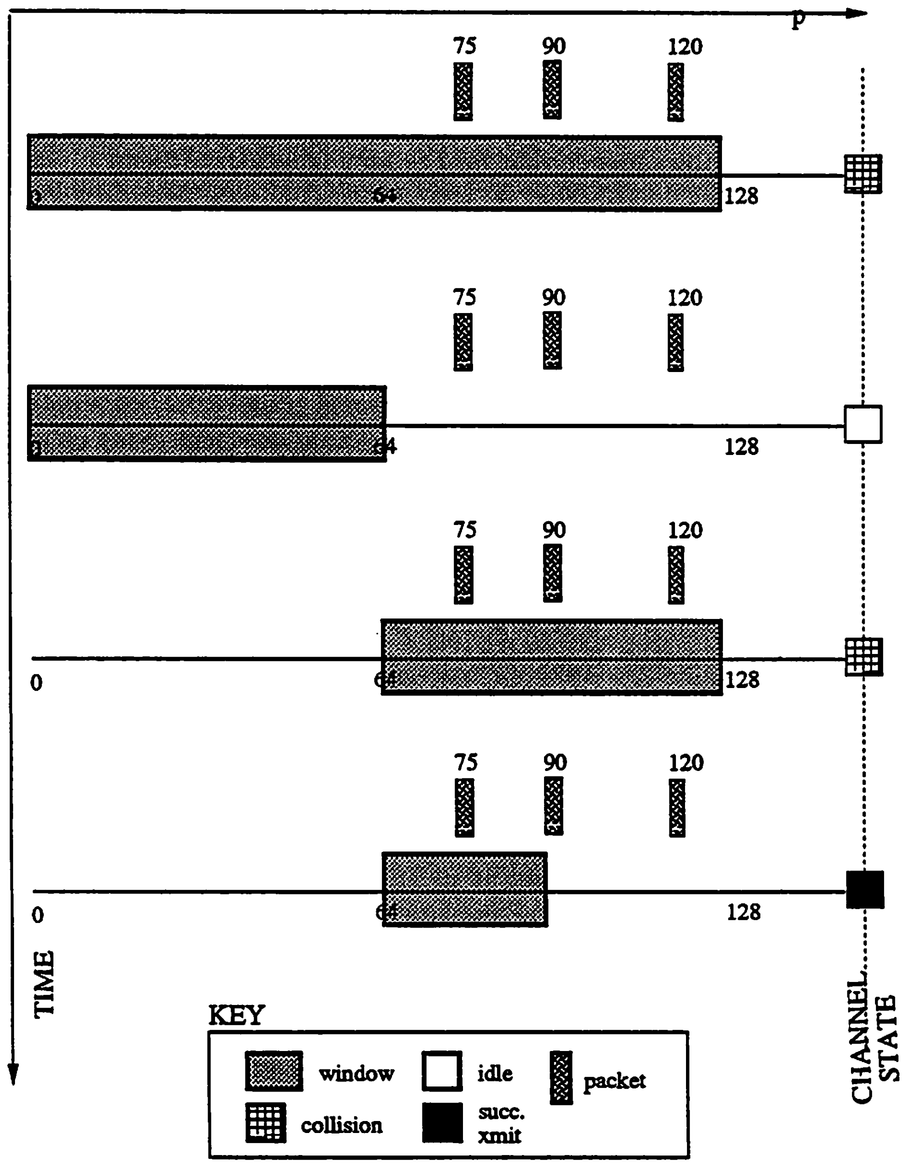

Figure 13 An example of the window protocol in operation

8.1 RTCOS/MAC: PRIORITY RESOLUTION PROTOCOL

Implementation of priority resolution using the above contention resolution approach is straightforward. Note that priority resolution is an instance of PBCR, where the parameter p associated with each node is the priority associated with the packet with the highest priority (assuming that higher priorities correspond to smaller p values) at the node[5]. It can be shown that the contention resolution overhead per packet for the priority resolution window protocol lies in the range $[0, 2\lceil \log_2 K \rceil - 1]$, where K is the number of priority levels. Note that the average overhead, given by $\lceil \log_2 K \rceil - 0.5$, increases logarithmically with K and is about the same as that for the forcing headers scheme.

8.2 RTCOS/MAC: REAL-TIME VIRTUAL CIRCUIT PROTOCOL

We have proposed a protocol known as the RTVC window protocol, based on the same window splitting approach, that can guarantee bounded channel access times and hence can be used to support the abstraction of real-time virtual circuits. This protocol assumes that the system consists of a fixed number (N_{rtvc}) of real-time virtual circuits, chosen *a priori*. Each real-time virtual circuit is characterized by a unique identifying integer known as its capability value. The packets at the heads of real-time virtual circuit transmission queues contend for channel access using the window protocol described above. The parameter p in the window protocol that is associated with a node is equal to the capability value of the enabled real-time virtual circuit with the smallest capability value (a real-time virtual circuit is said to be enabled, if it is permitted to contend for access to the channel; otherwise it is said to be disabled). A real-time virtual circuit is disabled, whenever a packet belonging to a real-time virtual circuit with a higher capability value is transmitted. It remains disabled until all real-time virtual circuits, with a capability value that is higher than that of the real-time virtual circuit that just transmitted a packet, have had a chance to transmit. These actions of enabling and disabling real-time virtual circuits ensure that the channel access time for each real-time virtual circuit is bounded. The reader is referred to (Arvind, Ramamritham and Stankovic 1990) for more details. The RTVC window protocol has a worst case channel access time given by $N_{rtvc}(P + \xi)$, where the overhead $\xi = (2\lceil \log_2 N_{rtvc} \rceil - 1)\tau$ and τ is the round trip propagation delay. This is slightly larger than the worst case channel access time of $N_{rtvc}(P + \tau)$ for some of the other protocols that we saw in Section 7.1.2. The fractional increase in the worst case channel access time because of using the RTVC window protocol is given by $\frac{2\alpha(\lceil \log_2 N_{rtvc} \rceil - 1)}{1+\alpha}$, where α is the ratio of the round trip propagation delay to the packet size. Thus the fractional increase in the worst case channel access time for a system with 32 real-time virtual circuits and $\alpha = 0.01$ is about 8 percent; for a system with 1024 real-time virtual circuits the increase is about 18 percent (note the logarithmic relationship between the number of real-time virtual circuits and the worst case contention resolution overhead). However, results of a simulation study in (Arvind, Ramamritham and Stankovic 1990) indicate that the performance of the RTVC window protocol (measured in terms of the guarantee ratio, the fraction of the number of packets that arrive that get guaranteed) is close to that of an idealized protocol

[5]A similar protocol is specified in the IEEE 802.4 token bus standard for station addition. However this protocol uses recursive quarternary partitioning, while our protocol uses recursive binary partitioning.

that can guarantee a bounded channel access time of $N_{rtvc}P$ with zero overhead. This is to be expected since the guarantee ratio is mainly dependent on the worst case channel access time. It 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.

8.3 RTCLS/MAC PROTOCOL

Implementation of real-time datagram arbitration based on the minimum laxity first policy using the above window splitting paradigm is again a simple matter. The parameter p associated with each node is made equal to the laxity of the packet with the smallest laxity awaiting transmission at the node. However, since laxity values need not be bounded, a maximum window size is chosen so that the window protocol will have a starting point. Packets whose laxities are greater than the maximum window size do not participate in the contention resolution process. Thus only the most "urgent" packets are considered for transmission at any instant of time. However since the laxity of a waiting packet continuously decreases with time, the laxity of every packet will eventually fall within the window. We have incorporated these ideas into a new window protocol for real-time datagram arbitration known as the RTDG window protocol. The results of simulation experiments indicate that the performance of the RTDG 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.

The RTDG protocol and the RTVC protocol have further been combined to implement two integrated window protocols known as INTPVC and INTPDG, that can be used to support both RTCLS and RTCOS on a common channel. In addition to the usual advantages of integrated protocols, including reduced costs resulting from common interfaces and cabling, and efficient bandwidth utilization, these protocols also display an improvement in the quality of support for RTCLS and/or RTCOS over a range of system parameters. The protocol INTPVC accords greater importance to servicing real-time virtual circuit packets; real-time datagrams are considered for transmission, only if there are no real-time virtual circuits with packets pending for transmission. The protocol INTPDG accords greater importance to real-time datagram services; real-time datagrams are considered for transmission even if there are real-time virtual circuit packets pending for transmission (but in a manner that does not violate the bounded channel access time property of real-time virtual circuits). The reader is referred to (Arvind, Ramamritham and Stankovic 1990) for more details on these protocols.

In this section, we described a suite of window protocols that represents an attempt to support both RTCOS and RTCLS in a uniform manner. In addition to providing the advantage of homogeneity (enabling the use of a uniform medium access control logic and LAN controller hardware to support both RTCOS and RTCLS), these protocols also have performance characteristics that are close to the best available (or ideal) protocols that can be used to provide the same functionality.

9 *CONCLUSION*

Most current work in real-time communication deals with static systems in which messages with timing constraints are mainly periodic, or occur only rarely. However the dynamic closed loop distributed real-time systems of the future will be characterized by richer communication patterns. Specialized network services and protocols will be required to support the communication requirements of these systems. In this paper, we proposed RTLAN, a new local area network architecture for communication in such systems. RTLAN provides new classes of connection-oriented and connectionless services known as RTCOS and RTCLS respectively, that take the timing constraints of messages explicitly into account. In order to provide these services, RTLAN employs specialized real-time medium access control protocols. We presented several medium access control protocols that can be used at the MAC layer of RTLAN. Finally, we also described a homogeneous set of MAC protocols that may be used to support both RTCOS and RTCLS using a uniform window splitting paradigm.

ACKNOWLEDGEMENTS

We would like to thank Erich Nahum, Douglas Niehaus, David Yates and Goran Zlokapa for their constructive comments and criticisms on an earlier version of this paper.

BIBLIOGRAPHY

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

[2] Alger, L.S., Lala, J.H., A Real-Time Operating System for a Nuclear Power Plant Computer, *Proc. Real-Time Systems Symposium*, December 1986.

[3] Arvind, K., Ramamritham, K., Stankovic, J.A., Window Protocols for Real-Time Communication Services, *Submitted for Publication*, January 1991.

[4] Bares, J., Hebert, M., Kanade, T., Krotkov, E., Mitchell, T., Simmons, R., Whittaker, W., Ambler, An Autonomous Rover for Planetary Exploration, *Computer*, Vol. 22., No. 6, June 1989, pp. 18-26.

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

[6] Bihari, T.E., Walliser, T.M., Patterson, M.R., Controlling the Adaptive Suspension Vehicle, *Computer*, Vol. 22., No. 6, June 1989, pp. 59-65.

[7] 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.

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

[9] Comer, D., Yavatkar, R., FLOWS: Performance Guarantees in Best Effort Delivery Systems, *IEEE INFOCOM'89*.

[10] Cristian, F., Synchronous Atomic Broadcast for Redundant Broadcast Channels, *Real-Time Systems*, Vol. 2, No. 3, September 1990, pp. 195-212.

[11] Cristian, F., Dancey, R.D., Dehn, J., Fault-Tolerance in the Advanced Automation System, *IBM Research Report* RJ 7424 (69595), April 1990.

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

[13] *IEEE Standards for Local Area Networks. Token-Passing Bus Access Method*, IEEE, New York, 1982.

[14] *IEEE Standards for Local Area Networks: Logical Link Control*, IEEE, New York, 1984.

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

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

[17] 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.

[18] Iyengar, S.S., Kashyap, R.L., Autonomous Intelligent Machines, *Computer*, Vol. 22, No. 6, June 1989.

[19] Jacobs, I.M., Binder, R., Hoversten, E.V., General Purpose Packet Satellite Networks, *Proceedings of the IEEE*, 1978, pp. 1448-1468.

[20] 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.

[21] 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.

[22] 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.

[23] Knightson, K.G., Knowles, T., Larmouth, J., *Standards for Open Systems Interconnection*, McGraw-Hill Book Company, New York, 1988.

[24] Kopetz, H., Real Time in Distributed Real Time Systems, *Proc. IFAC Distributed Computer Control Systems*, 1983.

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

[26] 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.

[27] Lark, J.S., Erman, L.D., Forrest, S., Gostelow, K.P., Hayes-Roth, F., Smith, D.M., Concepts, Methods, and Languages for Building Timely Intelligent Systems, *The Journal of Real-Time Systems*, Vol. 2, 1990, pp. 127-148.

[28] LeBlanc, T.J., Markatos, E.P., Operating System Support for Adaptable Real-Time Systems, *Proc. Seventh IEEE Workshop on Real-Time Operating Systems and Software*, May 1990, pp. 1-10.

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

[30] 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.

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

[32] 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.

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

[34] 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.

[35] 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.

[36] Muratore, J.F., Heindel, T.A., Murphy, T.B., Rasmussen, A.N., McFarland, R.Z., Real-Time Data Acquisition at Mission Control, *Communications of the ACM*, Vol. 33, No. 12, December 1990, pp. 18-31.

[37] Nielsen, K., *Ada in Distributed Real-Time Systems*, McGraw-Hill, New York, 1990.

[38] 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.

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

[40] 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.

[41] 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.

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

[43] Rouse, W.B., Geddes, N.D., Hammer, J.M., Computer-Aided Fighter Pilots, *IEEE Spectrum*, Vol. 27, No. 3, March 1990.

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

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

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

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

[48] 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.

[49] Tanenbaum, A.S., *Computer Networks*, Prentice-Hall, 1989.

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

[51] 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.

[52] Weisbin, C.R., de Saussure, G., Einstein, J.R., Pin, F.G., Heer, E., Autonomous Mobile Robot Navigation and Learning, *Computer*, Vol. 22, No. 6, June 1989, pp. 29-35.

[53] Whittaker, W.L., Kanade, T., Japan Robotics Aim for Unmanned Space Exploration, *IEEE Spectrum*, Vol. 27, No. 12, December 1990, pp. 64-67.

[54] 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.

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