Real Time Systems, 9, 75–107 (1995)

Hard Real-Time Communication in Multiple-Access Networks

NICHOLAS MALCOLM AND WEI ZHAO {MALCOLM, ZHAO}@CS.TAMU.EDU
Department of Computer Science
Texas A&M University
College Station
TX 77843-3112

Editor: John A. Stankovic

Abstract. With the increasing use of distributed hard real-time systems, the ability of computer networks to handle hard real-time message traffic is becoming more important. For traditional networks, maximizing the throughput or minimizing the average message delay is the most important performance criteria. In the hard real-time domain, however, concern focuses on satisfying the time constraints of individual messages. This paper examines recent developments in hard real-time communication in local area multiple-access networks. Two general strategies are used in hard real-time communication: the guarantee strategy and the best-effort strategy. In the former, messages are guaranteed to meet their deadlines during normal operation of the network. In the best-effort strategy, the network will attempt to send messages before their deadlines, but no guarantees are given. Real-time message traffic can be distinguished according to whether it is best suited for the guarantee strategy or the best-effort strategy. Although this paper concentrates on multiple-access networks, many of the concepts presented and lessons learned are also applicable to other types of networks.

Keywords: Media access control (MAC), multiple-access networks, real-time communication.

1. Introduction

The use of computers in applications such as embedded command and control systems, image processing and transmission, integration of expert systems into avionics, automated manufacturing, and industrial process control is becoming increasingly common. These applications are often implemented as distributed computer systems. This is due not only to the nature of the applications, but also to the potential that distributed systems have for providing good reliability, good resource sharing, and good extensibility (Kleinrock 1985; Stankovic 1984; Yau 1989).

The key to success in using a distributed system for applications such as those mentioned above is the timely execution of computational tasks that usually reside on different nodes and communicate with one another to accomplish a common goal. The correctness of the system depends on both the logical results and the time at which those results appear (Krishna and Lee 1991; Sha 1994; Shin 1987; Stankovic 1988; Stankovic and Ramamritham 1988, 1993; van Tilborg and Koob 1991a, 1991b; Zhao 1989). It is difficult to ensure timely computational results in such systems without a network that supports the timely delivery of inter-task messages.

This paper discusses hard real-time communication in local area multiple-access networks. In *hard* real-time communication, messages have explicit time constraints such as deadlines. A message must satisfy its time constraints or it is considered to be lost. Local area multiple-access networks typically have a geographical expanse covering a building or a group of buildings, or a single ship or aircraft. They are one of the most common types of networks used to support distributed hard real-time applications.

To meet their time constraints, hard real-time messages must be properly scheduled for transmission. Scheduling messages in a multiple-access network is the function of the Media Access Control (MAC) protocol. It is this protocol that arbitrates access to the network and determines what message to transmit at any given time. Although many factors affect the performance of a network over which real-time messages are transmitted, in this paper we concentrate on MAC protocols because they have a significant impact on the real-time performance of a network.[1]

There have been many studies dealing with MAC protocols for multiple-access networks (Stallings 1991; Tanenbaum 1988). These studies typically focus on issues such as maximizing the message throughput or minimizing the average message delay. In contrast, a MAC protocol for hard real-time message traffic must consider the timing constraints of *individual* messages. The most important design consideration is to ensure that message deadlines are met or that the number of missed deadlines is minimized. An excellent survey of earlier work can be found in (Kurose et al. 1984). We concentrate mainly on results since the mid 1980s.

Two main strategies are used in sending real-time messages. In the *guarantee strategy*, an attempt is made to guarantee ahead of transmission time that the real-time messages will meet their deadlines. The guarantee may be given either during system design time or during system operation. The key feature is that once a message is accepted for transmission, it is guaranteed to meet its deadline. The ability to guarantee that messages will meet their deadlines is important in distributed hard real-time systems. The provision of guaranteed communications is an integral part of designing mission critical systems in which real-time tasks on different nodes communicate to achieve a common goal.

The other strategy used for real-time communication is the *best-effort* strategy. In the best-effort strategy, the network will try to meet the message deadlines, but no guarantees are given. The best-effort strategy is used when there are insufficient network resources to meet all message deadlines. The best-effort strategy can also be used when the applications involved (e.g., digital voice) can tolerate a certain amount of message loss. Allowing some message loss can enable the network to support more message traffic by taking advantage of statistical multiplexing.

Although this paper concentrates on multiple-access networks, many of the concepts presented and lessons learned are equally applicable to other types of networks. For example, the concept of providing both guaranteed and best-effort services for real-time message traffic has also been proposed for integrated services packet networks (Clark et al. 1992).

One of the key lessons learned in designing real-time communication protocols for multiple-access networks is that network scheduling is fundamentally different from centralized processor scheduling. Much of the work on network scheduling has been based on existing centralized scheduling algorithms. However, network scheduling is distributed in nature, and substantial overhead is usually required to obtain accurate global state information. Partly because of this overhead, optimality results for centralized scheduling do not always carry over to network scheduling. The benefits of using a good centralized scheduling algorithm can be nullified in the distributed case if the overhead is too high. Hence, a recurring theme in the design of hard real-time MAC protocols is to use minimal overhead in emulating a centralized scheduling policy. This means that nodes may often make scheduling decisions with only imperfect knowledge about the status of other nodes in the network.

The remainder of this survey is organized as follows. Section 2 defines the system and message models used in hard real-time communication. Sections 3 and 4 discuss approaches for scheduling synchronous and asynchronous messages, respectively. Section 5 contains a summary and a brief discussion of related areas of work.

2. Models

This paper concentrates on multiple-access networks. In a multiple-access network, nodes communicate via a single shared channel. Nodes in a multiple-access network usually have a layered communication architecture. Figure 1 shows the layered architecture of the ISO Reference Model for Open Systems Interconnection (OSI) (Zimmermann 1981). Each layer has a different set of protocols responsible for carrying out the functions required of the layer. For example, the protocols in the data link layer are responsible for providing reliable transmission of data frames across a communication link. A *media access control* (MAC) protocol is responsible for selecting and sending messages over the shared channel of a multiple-access network. In terms of the OSI Reference Model, MAC protocols form part of the data link layer (Figure 1). Before we discuss MAC protocols and real-time messages in detail, we first consider the various kinds of delays that are experienced by a message when it is sent from one application to another. An understanding of these delays is important in considering whether the deadlines of messages will be met.

2.1. Message Delays

When using a multiple-access network to support distributed hard real-time systems, the application-to-application delay is crucial in determining whether application deadlines will be satisfied. The *application-to-application delay* is the time delay experienced by a message that is sent between application tasks. The application tasks often reside on different nodes, and so the message must be sent across the network.

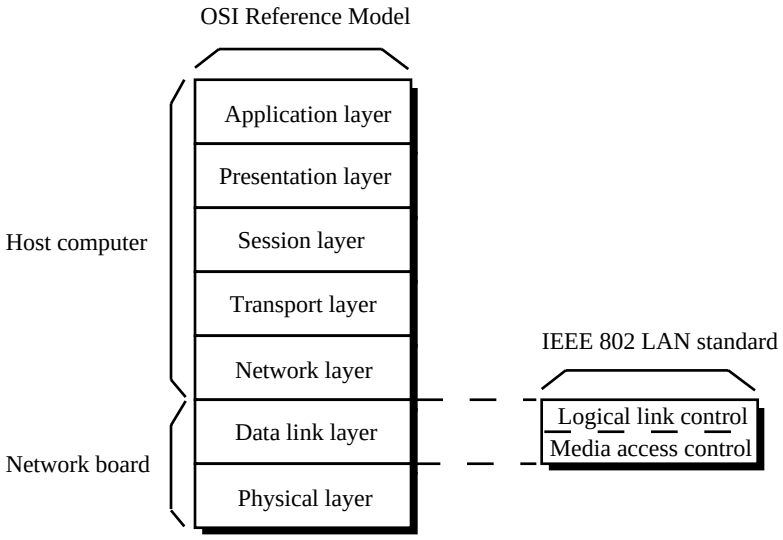

Figure 1. The OSI Reference Model

The application-to-application delay depends on the implementation of the system. In the majority of multiple-access networks to date, protocols in layers above the MAC protocol are realized in the host processor where applications are executed, while the MAC protocol is implemented by special network hardware — in the so called "network board." With this kind of system implementation, the application-to-application delay experienced by a message M can be decomposed as follows (Figure 2):

- A processing and queueing delay in the upper layers of the sending node. The processing delay includes the time required to create message headers. The queueing delay includes both the time during which a message is waiting to be processed, and the time during which a message is waiting to be passed to lower layer protocols. Let $d_{up\_send}(M)$ denote the maximum delay that may be experienced by message M in the upper layers of the sending node.

- A queueing delay in the MAC layer of the sending node. This delay occurs when a message is waiting to be sent by the MAC protocol. Let $d_{MAC\_send}(M)$ denote the maximum queueing delay that message M may experience in the MAC layer of the sending node.

- The message transmission delay. This delay is the time required to physically transmit the message. Let C_M be the transmission delay for message M.

- The propagation delay. This delay is the time required for a single bit to travel through the channel to the receiving node. Let this delay be denoted by τ.

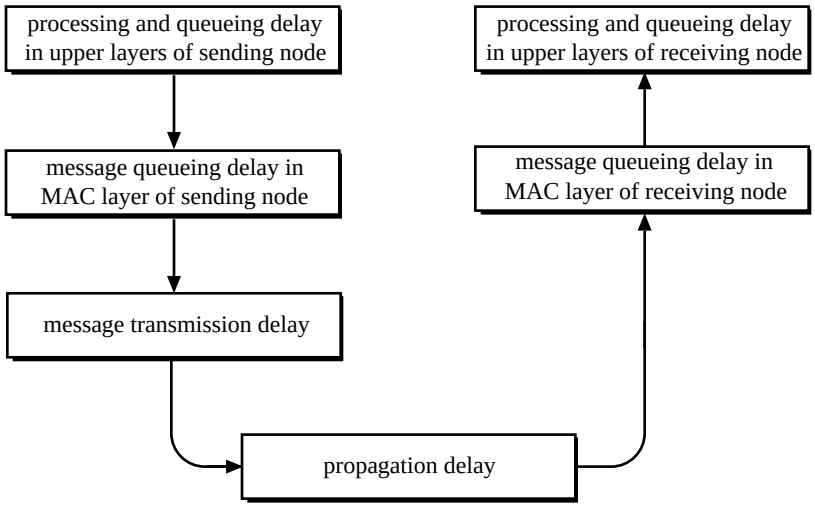

Figure 2. Decomposition of Application-to-Application Delay

- A message queueing delay in the MAC layer of the receiving node. Once a message is received at its destination, it is stored in a queue at the MAC layer of the receiving node until the message can be passed to the upper layer protocols. Let $d_{MAC\_receive}(M)$ denote the maximum delay that may be experienced by message M in the MAC layer of the receiving node.
- A processing and queueing delay in the upper layers of the receiving node. After a message is passed to the upper layer protocols, it may again be queued, before the message header is removed and the data passed to the receiving application task. Let $d_{up\_receive}(M)$ denote the maximum delay that may be experienced by message M in the upper layers of the receiving node.

An upper bound on the application-to-application delay for message M, $d_{app}(M)$, can now be written as

$$d_{app}(M) = d_{up\_send}(M) + d_{MAC\_send}(M) + C_M + \tau + d_{MAC\_receive}(M) + d_{up\_receive}(M). \quad (1)$$

Several of the components of the application-to-application delay are not determined by the network but by the hosts. The processing and queueing delays in the upper layer protocols ($d_{up\_send}(M)$ and $d_{up\_receive}(M)$) depend mainly on the system software and the processor/memory speeds at the sending and receiving hosts. The queueing delay in the MAC layer of the receiving node ($d_{MAC\_receive}(M)$) depends on how quickly a host processor can respond to a "message arrived" interrupt from the MAC layer. For a given system, an upper bound on these delays can often be determined (Klein et al. 1994; Tindell 1993).

The message transmission delay (C_M) is given by the bandwidth of the channel, and the message propagation delay (τ) is given by the signal propagation speed. Both of these delays can be regarded as fixed. However, the queueing delay in the MAC layer of the sending node ($d_{MAC\_send}(M)$) depends on the MAC protocol that is used. An appropriately chosen MAC protocol can reduce the delay of messages with urgent time constraints (possibly resulting in larger delays for other messages) and thus help real-time messages to meet their deadlines. Because of the impact that the choice of MAC protocol has on the real-time performance of a network, we concentrate on MAC protocols in this paper.

2.2. MAC Protocols

Scheduling message transmissions over a channel is the responsibility of the MAC protocol. For the purpose of understanding its real-time performance, it is convenient to consider a MAC protocol as consisting of two processes: an *access arbitration* process and a *transmission control* process. The access arbitration process determines *when* a node can send a message over the channel. The transmission control process determines *how long* a node can continue to send messages over the channel.

To illustrate the role of these processes, consider the Time Division Multiple-Access (TDMA) protocol. In this protocol, nodes in the network are arranged into a logical ring. When a node has finished using the channel, the access arbitration process gives channel transmission rights to the downstream neighbor of the node that has just finished using the channel. This can be implemented by passing a token to the downstream neighbor. When a node is given permission to begin sending messages, the transmission control process allows the node to transmit messages for H time units. H is a fixed protocol parameter that is common to all nodes. Upon expiration of H time units, the access arbitration process again passes control to the next node in the ring.

There are many other possible access arbitration processes beyond the simple one described above. For example, the access arbitration process in the IEEE 802.5 protocol (IEEE 1989) allows network-wide priority arbitration to be performed. Other access arbitration processes such as those studied in some sliding window protocols (Kurose 1984) use the past history of channel activity to decide which node should send a message next. The access arbitration process often has a distributed implementation, and is not necessarily physically resident on any one single node.

The choice of access arbitration process is important because it affects issues such as fairness and metrics such as the throughput and the message delay. Further, for real-time communication, there are additional points to be addressed:

- *Timing correctness.* In order to satisfy the time constraints of hard real-time messages, the next node to gain access to the channel must be chosen correctly. For example, giving precedence to a node with a message whose time constraints

are not tight could lead to the loss of a tightly constrained message on another node.

- *Overhead.* In a network environment, nodes usually have little knowledge about the messages waiting on other nodes. Consequently, the access arbitration process may often make its decision based on incomplete or out-of-date information. In order to offset this problem, the access arbitration process may attempt to gather information about the state of other nodes. However, the extra overhead incurred in gathering information about other nodes results in less channel bandwidth being available to normal message traffic.

In contrast to access arbitration processes, comparatively little work has been done on transmission control processes. One common approach is to allow a node to send only a single message. The access arbitration process will then give another node access to the channel. Another approach is to allow a node to send any number of messages before control passes to another node. In general there are two classes of transmission control processes. In a *static* transmission control process, the amount of time that a node may continue to send messages over the channel is fixed at design time. In a *dynamic* transmission control process, the amount of send time is determined during network operation.

The operation of the transmission control process has a large impact on whether the time constraints of messages will be met. The time constraints of messages cannot be met predictably without a guaranteed minimum amount of time in which a node can uninterruptibly send messages. On the other hand, allowing a node to monopolize the channel for too long can result in message time constraints being violated on other nodes in the network.

The access arbitration and transmission control processes will be discussed in more detail in Sections 3 and 4. Ideally, any MAC protocol for hard real-time communication should combine both an appropriate access arbitration process and an appropriate transmission control process. Much of the existing work, however, has tended to concentrate more on one process than the other. For example, as will be seen in Section 3, the work on synchronous message transmission with the IEEE 802.5 protocol has emphasized the access arbitration process. The work on synchronous bandwidth allocation for the timed token protocol, however, has emphasized the transmission control process.

2.3. Message Classification

Hard real-time messages fall into two categories: *synchronous* messages and *asynchronous* messages.[2] A distributed hard real-time system will typically contain both types of messages, and we will now discuss them both in turn.

2.3.1. Synchronous Messages

Synchronous messages are used for transmitting sensory data or for intertask communication between periodic tasks. We assume that there are n streams of synchronous messages $S_1, \ldots, S_n$ in the network. A *synchronous message stream* S_i consists of a sequence of messages with constant interarrival times. Messages in a synchronous message stream are referred to as synchronous messages. The *period* T_i of stream S_i is the interarrival time of messages in stream S_i. Each message M in a synchronous message stream has the following characteristics:

- An *arrival time*, which is the time at which the message becomes available for transmission. The arrival time of a synchronous message is the beginning of a period for its synchronous message stream.
- A *length*, which is the number of time units required to transmit the entire message.
- A *MAC layer deadline*, D_M, which defines a time constraint for the transmission of message M. If message M with deadline D_M arrives at the MAC layer of the sending node at time t, then the sending node must finish transmitting the message by time $t + D_M$.

  As discussed in Section 2.1, a message experiences several delays en route to its destination. Satisfying a message deadline is directly related to these delays. Specifically, the MAC layer deadline of message M can be met if

  $$D_M \geq d_{MAC\_send}(M) + C_M, \tag{2}$$

  where $d_{MAC\_send}(M)$ is the maximum queueing delay that message M may experience in the MAC layer of the sending node, and C_M is the transmission time of message M, as defined in Section 2.1. Note that C_M is fixed for a given network. Different MAC protocols may result in different values of $d_{MAC\_send}(M)$, which affects the ability of the different protocols to meet message deadlines.

  In a real system, message time constraints may also be given as application layer deadlines. The *application layer deadline* of a message M, denoted $D_{M_{app}}$, gives the maximum amount of time that may elapse before the message must be received by the application task to which it is sent. Using the delay bounds introduced in Section 2.1, the MAC layer deadline of message M may be expressed in terms of the application layer deadline as follows:

  $$D_M = D_{M_{app}} - d_{up\_send}(M) - \tau - d_{MAC\_receive}(M) - d_{up\_receive}(M). \tag{3}$$

  Thus, (3) gives us a way to convert an application layer deadline into a MAC layer deadline. This paper concentrates on the MAC layer. Hence, in the rest of the paper, whenever we refer to deadlines we are referring to MAC layer deadlines.

Given a set of synchronous message streams with periods $T_1, T_2, \ldots, T_n$ and message lengths $C_1, C_2, \ldots, C_n$, the *utilization* of the synchronous message set is defined as

$$U_S = \sum_{i=1}^{n} \frac{C_i}{T_i}. \tag{4}$$

The utilization U_S represents the proportion of time that will be needed for transmitting this set of synchronous message streams.

2.3.2. Asynchronous Messages

Asynchronous messages are often used to carry alert information or for communication between aperiodic tasks. In contrast to synchronous messages, asynchronous messages have stochastic interarrival times. Prior to their arrival the system may have little knowledge of their timing characteristics. Like synchronous messages, asynchronous messages may be associated with an arrival time, a length, and a deadline D_M. Another time constraint that is often used for an asynchronous message M is the laxity of the message at time t, denoted $LA_M(t)$. The laxity of a message M is defined as the amount of time remaining before transmission of the message must begin if the message is to meet its deadline. That is, if message M arrives at time t_{arr}, we have

$$LA_M(t) = t_{arr} + D_M - C_M - t \tag{5}$$

where C_M is the length of the message.

If the average arrival rate of asynchronous messages to a network is λ, and the average length of asynchronous messages is C, then the demand utilization U_A of the asynchronous messages is defined as

$$U_A = \lambda * C. \tag{6}$$

If some asynchronous messages are not transmitted due to missed deadlines, then the actual utilization may be lower than the demand utilization.

3. Communication with Synchronous Messages

Synchronous messages arrive periodically and must satisfy their deadlines or they are considered to be lost. Because synchronous messages are deterministic — all of their timing characteristics are known in advance — most of the research in this area has followed the guarantee strategy. The goal has been to guarantee that all synchronous messages will meet their deadlines during error-free operation of the network. In some applications, it is acceptable for synchronous messages to miss their deadlines occasionally (Schulzrinne et al. 1990). However, the design of multiple-access protocols to take advantage of this is still an open area of research.

Recall that a MAC protocol consists of both an access arbitration and a transmission control process. In order to guarantee that message deadlines will be met, both of these processes must be carefully chosen. Most of the existing work on guaranteeing synchronous messages, however, has tended to emphasize one process over the other. This has given rise to the following two approaches for guaranteeing that synchronous messages will meet their deadlines:

1. *The access arbitration based approach.* This approach concentrates on the use of an appropriate access arbitration process. A transmission control process is, of course, still used with this approach, but the emphasis lies on using the access arbitration process to guarantee synchronous message deadlines. The best example of the access arbitration based approach is rate monotonic scheduling with priority driven protocols (Pleinevaux 1992; Strosnider 1988; Strosnider et al. 1988, 1989). In a priority driven protocol, each message is assigned a priority and the access arbitration process tries to ensure that higher priority messages are sent before lower priority messages. The rate monotonic algorithm attempts to assign priorities to messages in such a way that synchronous message deadlines are guaranteed.

2. *The transmission control based approach.* This approach concentrates on using a static transmission control process to meet synchronous message deadlines. We will focus on using this approach with the timed token protocol. In the timed token protocol, the transmission control process allows each node i a fixed amount of time H_i (called the synchronous bandwidth) in which to send synchronous messages upon gaining transmission rights. Synchronous messages can be guaranteed by setting the synchronous bandwidths appropriately.

We concentrate on rate monotonic scheduling and synchronous bandwidth allocation to illustrate the access arbitration based and transmission control based approaches, respectively. This is because there has been thorough and extensive work in these areas since the mid 1980s. Further, the protocols involved, i.e., both the priority driven and the timed token protocols, are used in current network standards. For example, the IEEE 802.5 standard (IEEE 1989) uses a priority driven protocol, and the timed token protocol is employed in the FDDI standard (ANSI 1990). We will also briefly discuss other examples of the access arbitration based and transmission control based approaches. A more in depth discussion can be found in (Malcolm and Zhao 1994).

Regardless of the approach taken, a protocol that can guarantee synchronous messages should exhibit the following properties:

1. *Synchronous message utilization.* A protocol has an achievable utilization $\mathcal{U}$ if it can guarantee all synchronous messages from streams whose total utilization is no more than $\mathcal{U}$. The *worst case achievable utilization* of a protocol is the least upper bound of its achievable utilizations. Provided that the utilization of a set of synchronous message streams is less than the worst case achievable utilization, the synchronous messages are guaranteed to meet their deadlines. A

message set whose deadlines are guaranteed to be met is known as a schedulable message set.

At system design time, the exact characteristics of the synchronous message sets that will use the system might not be known. If a choice must be made regarding which MAC protocol to use in the system, the *average breakdown utilization* can be of interest. The average breakdown utilization of a protocol is defined as the average utilization U_S of a schedulable message set such that increasing some of the message lengths would result in deadlines being missed (Kamat and Zhao 1993; Lehoczky et al. 1989; Strosnider and Katcher 1991). Intuitively, the average breakdown utilization reflects how high a message set utilization can be on average without missing some message deadlines. It is preferable to choose a MAC protocol that has a high average breakdown utilization.

2. *Robustness.* The protocol should be robust in the face of changes to the characteristics of synchronous message streams. A small change in the characteristics of the synchronous message streams should not affect their overall schedulability, and a change in the streams originating at a given node should affect only that node.

3. *Limited propagation of timing faults.* If the messages in a synchronous message stream are longer than specified or arrive more frequently than specified, then the synchronous message stream violates its specifications. This can disrupt the predicted operation of the network, resulting in deadlines being missed. It is desirable that such a violation of specifications should not affect the other streams in the network; messages from other streams should still meet their deadlines.

4. *Asynchronous message handling.* In practice, a network for distributed hard real-time systems may handle asynchronous as well as synchronous messages. Even if asynchronous messages are not time constrained, it is important that they have a high throughput in order to avoid problems such as buffer overflow. If asynchronous messages are time constrained, then the protocol should attempt to maximize the proportion of asynchronous messages that satisfy their time constraints. If loss of asynchronous messages is not permissible, then techniques such as using a periodic server for asynchronous messages may be used (Section 4.1.1). Provided that the network is not overloaded, a periodic server can guarantee that all of the asynchronous messages will meet their deadlines.

5. *Runtime overhead.* The runtime overhead of a protocol includes the time spent by the access arbitration process in arbitrating access to the communication channel. If two protocols give the same schedulability performance for real-time messages, the protocol with the lower runtime overhead is preferred because more channel bandwidth will be available to accommodate future changes in the system.

Until recently, much of the work on synchronous messages in multiple-access networks has assumed that message deadlines D_M are equal to the period length.

Consequently, much of the work reported in the remainder of this section uses this assumption.

3.1. The Access Arbitration Based Approach

We now consider the use of the access arbitration process in guaranteeing synchronous message deadlines. In particular, we will focus on using the access arbitration process to implement the rate monotonic scheduling algorithm (Strosnider et al. 1988, 1989).

An essential issue in the operation of an access arbitration process is deciding which node should use the channel next. This is especially critical in real-time systems, because a wrong choice can result in messages missing their deadlines. The operation of an access arbitration process is analogous to the operation of a task scheduler on a processor. Instead of deciding which node should use the channel next, a task scheduler decides which task should use the processor next. Because of the similarity between processor and network scheduling and the large amount of work that has already been done on processor scheduling, there has been considerable interest in adapting existing processor scheduling algorithms for use in networks. Strosnider et al. (1988, 1989) have proposed using the rate monotonic processor scheduling algorithm in multiple-access networks. We now discuss this work in more detail.

3.1.1. The Rate Monotonic Algorithm

The rate monotonic algorithm of Liu and Layland (1973) is a static scheduling algorithm for scheduling independent periodic tasks on a uniprocessor. The rate monotonic algorithm assigns priorities to tasks in inverse proportion to their periods. Tasks with a smaller period are assigned a higher priority. The deadline of a task is assumed to be the end of the period in which the task arrived.

The rate monotonic algorithm is optimal in the sense that if any fixed priority preemptive scheduling algorithm can schedule a set of periodic tasks on a single processor, then so can the rate monotonic algorithm. Liu and Layland further show that the worst case achievable utilization of the rate monotonic algorithm for a set of n periodic tasks is $n(2^{1/n} - 1)$. For large n, this approaches $\ln 2 \approx 0.69$. If the total utilization of a set of tasks is less than 0.69, then the set of tasks can always be scheduled by the rate monotonic algorithm.

The rate monotonic algorithm is attractive because it exhibits the following desirable properties. It has a high worst case achievable utilization. It is robust in the sense that the characteristics of a task set may be freely modified and the tasks will still be schedulable, provided that the total utilization of the tasks remains less than 0.69. It can easily handle aperiodic (asynchronous) tasks; aperiodic tasks are scheduled to run as they arrive, up until the point that periodic tasks would begin to miss their deadlines. It has a relatively low runtime overhead, because

task priorities are assigned during system design time. At run time it only remains to preempt low priority tasks upon arrival of a high priority task.

3.1.2. Use of the Rate Monotonic Algorithm in a Multiple-Access Network

The optimality and robustness of the rate monotonic algorithm make it an attractive choice for scheduling the transmission of synchronous messages. The implementation of the rate monotonic algorithm in a multiple-access network with a priority driven protocol has been considered by Strosnider et al. (Strosnider 1988; Strosnider et al. 1988, 1989). Synchronous messages with a smaller period are assigned a higher priority. Because the rate monotonic algorithm requires higher priority tasks to preempt lower priority tasks, a network protocol (i.e., the access arbitration process) that implements the rate monotonic algorithm must preempt the transmission of low priority messages when a high priority messages arrives. This is approximated by dividing messages into packets. The global priority arbitration provided by the access arbitration process then ensures that the highest priority packet is always sent first (in the absence of new message arrivals during access arbitration). The packet size used in the system is a tradeoff between the higher preemptability of small packets and the higher throughput of large packets. Using small packets allows a newly arrived high priority message to preempt a lower priority message sooner. However, large packets can give a higher throughput because the ratio of protocol overhead to packet payload data is lower.

3.1.3. Implementation of Priority Arbitration

In order to use the rate monotonic algorithm in a network, network-wide priority arbitration must be provided. It is important that the access arbitration process have a low runtime overhead, otherwise the achievable utilization of the network may be low because there is less time available for transmitting messages.

The two most common types of multiple-access networks are ring networks and bus networks. The following discussion gives some simple protocols for implementing global priority arbitration in these types of multiple-access networks. Note that in any such protocol, priority arbitration itself will require some time to complete. If new messages arriving during this time are not considered in the current round of priority arbitration, then the protocol will not implement the exact highest-priority-first policy.

Ring Networks

In a ring network (Liu 1978), nodes are physically arranged in a ring. A special bit pattern called the *token* circulates among the nodes and is used to control access to the ring. A node wishing to send a packet must first gain possession of the token,

and then transmit the packet. After transmitting the packet, the node releases the token back into the ring. The access arbitration process is responsible for the circulation and seizure of the token. The transmission control process determines how long a node may transmit packets before it is required to release the token.

To implement global priority arbitration, the token is used to arbitrate access to the channel in such a way that higher priority packets have precedence. The token contains a priority field, and a node may only seize the token and transmit a packet if the priority of the packet is greater than or equal to the priority field in the token. An example of a token passing protocol that provides global priority arbitration is the IEEE 802.5 protocol (IEEE 1989). A brief outline of this protocol is given below:

- When a node sends a packet, it places the token in *reservation* mode and appends the packet to the token. No other node can transmit a packet when the token is in reservation mode. However, when the token is in reservation mode, any node with a packet whose priority is higher than the priority field in the token will write the packet priority into the token.
- When a token in reservation mode returns to the node that is sending the packet, the token is placed in *free* mode and released back into the ring. The priority of the free token will be the maximum of the priority field in the reservation token and the highest priority pending packet on the sending node.
- Any node with a packet to send may now seize the free token and send the packet, provided that the priority of the packet is greater than or equal to the priority field in the free token.

Bus Networks

There are two different kinds of bus networks: those used as backplane buses and those used in local area multiple-access networks. A backplane bus is typically used to connect a processor with components such as main memory and DMA units. A set of wires is provided for data transfer, and another set of wires is provided to arbitrate access to the bus. For example, the Futurebus+ dedicates 8 pins for priority arbitration (Sha et al. 1990). These pins allow priority arbitration to occur concurrently with data transfers. Hence, while the current data transfer is in progress, the next bus master may be elected by the arbitration process.

A bus in a local area multiple-access network often consists of only a single cable that is connected to nodes using OR logic. If several nodes transmit simultaneously, the value read back from the bus is the logical OR of the values transmitted. Because there is only a single cable, it must cater both for packet transmission and for the priority arbitration performed by the access arbitration process. One way to accomplish this is to arrange the nodes into a logical ring and operate the bus network similarly to a ring network.[3]

Another approach, which can be used in networks with a low bandwidth, is to use a variation of the binary countdown protocol (Mok and Ward 1979; Tanenbaum 1988). In this protocol, time is divided into alternating intervals of priority arbitration and packet transmission. During priority arbitration, time is slotted into intervals of length τ, where τ is the end-to-end propagation delay of the bus. Nodes with a packet to send broadcast one bit of the priority of the packet per time slot, beginning with the most significant bit. A node that broadcasts 0 and reads back 1 drops out of contention.

For example, suppose there are three nodes in the system, each with a packet to send (Figure 3). Packet P_1 has priority 110, packet P_2 has priority 010, and packet P_3 has priority 100. During the first time slot of the priority arbitration interval, each node broadcasts the most significant bit of its packet's priority. Node 2 drops out of contention because it broadcasts 0 and reads back 1 from the bus, which means that at least one other node has a higher priority packet. During the second time slot, node 3 drops out of contention because it broadcasts 0 and reads back 1 from the bus. During the third time slot, node 1 gains possession of the bus. The transmission control process is then invoked at node 1 and permits the transmission of packet P_1.

After transmission of packet P_1, a new priority arbitration interval begins. Any newly arrived packets may now take part in the new round of priority arbitration. In practice, each node will have a unique ID that can be appended to the priority in order to resolve any priority ties.

3.1.4. Comments

Using the rate monotonic algorithm a priority driven protocol can result in the successful transmission of synchronous messages sets that have a relatively high utilization. However, the rate monotonic algorithm is only approximated when using priority driven protocols; it cannot be exactly implemented. There are several reasons for this:

- *Priority arbitration time.* Using the rate monotonic algorithm with a priority driven protocol requires global priority arbitration each time that a packet is sent. If newly arriving messages are not considered in an ongoing round of priority arbitration, then the message selected for transmission might not have the highest priority in the system.

- *Number of priority levels.* The amount of time required to complete priority arbitration increases with the number of priority levels supported by the network. Consequently, network protocols usually support only a small number of priority levels. When using the rate monotonic algorithm, message priorities are based on the periods of the corresponding synchronous message streams. If the number of distinct periods is greater than the number of available priority levels, then some of the messages with different periods must be mapped to the same priority level. This can result in *priority inversion*, in which a

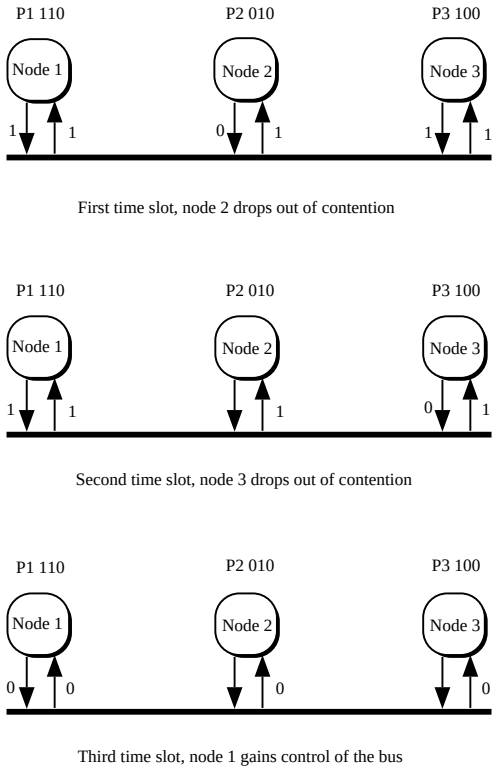

Figure 3. Priority Arbitration on a Bus

message with a theoretically lower priority is transmitted while a message with a theoretically higher priority is waiting (Tokuda et al. 1989). Lehoczky and Sha (1986) have shown how to allocate priorities so as to minimize the effect of priority inversion, but the problem cannot be completely eliminated.

- *Packet size.* In the analysis of the rate monotonic algorithm, it is assumed that tasks can be preempted by a higher priority task at any point in time. This assumption is reasonable in processor scheduling because a single CPU instruction requires only a negligible amount of time to execute. In a network, however, preemption cannot always occur immediately. Preemption must occur at packet boundaries.

Given that the rate monotonic algorithm can be approximately implemented in a multiple-access network, we now comment on several of the issues mentioned earlier: synchronous message utilization, robustness, timing faults, asynchronous message handling, and runtime overhead.

1. *Synchronous message utilization.* The worst case achievable utilization of the rate monotonic algorithm in priority driven networks is still an open area of research. However, by considering several factors, it is possible to find a lower bound on the worst case achievable utilization. After each interval of priority arbitration, a packet will be sent. We can account for the time required for priority arbitration by adding this time to the packet length (Strosnider et al. 1988). The length of a message can then be viewed as the sum of the lengths of its constituent packets (including the time for priority arbitration). Let B_{max} be the maximum time that the packets of a high priority message can be blocked by lower priority packets. This blocking time can arise either because the high priority message arrives during transmission of a low priority message or during an ongoing round of priority arbitration, or because of priority inversion due to an insufficient number of priority levels. Let P_{min} be a lower bound on synchronous message periods. The worst case achievable utilization, U^*, of the rate monotonic algorithm in priority driven networks satisfies

$$U^* \geq 0.69 - \frac{B_{max}}{P_{min}}. \tag{7}$$

   If the utilization of a synchronous message set (with the utilization being calculated using the augmented message lengths) is less than the above lower bound, then the synchronous message deadlines will be met.

   In addition to the worst case achievable utilization, the average breakdown utilization of the rate monotonic algorithm with priority driven protocols is also of interest (Kamat and Zhao 1993). Figure 4 shows the average breakdown utilization. The figure shows that the average breakdown utilization varies as the network bandwidth changes. The average breakdown utilization can be quite high. For example, when the bandwidth is around 10 Mbits/s, the average breakdown utilization is around 70%. A network bandwidth of 10 Mbits/s is typical for priority driven protocols such as the IEEE 802.5 protocol (IEEE 1989).

2. *Robustness.* If the characteristics of a synchronous message stream are changed, all messages will still meet their deadlines provided that the synchronous message utilization remains less than the worst case achievable utilization. As mentioned earlier, the exact value of the worst case achievable utilization for the rate monotonic algorithm in networks is still unknown. However, a lower bound on the worst case achievable utilization is given by (7).

3. *Timing faults.* We now consider the effects of a synchronous message stream violating its specifications. Such a violation occurs if messages have a shorter interarrival period or a longer length than specified. This can result in the deadlines of other streams being missed. However, the deadlines of synchronous messages with a higher priority than the offending stream are still guaranteed to be met.

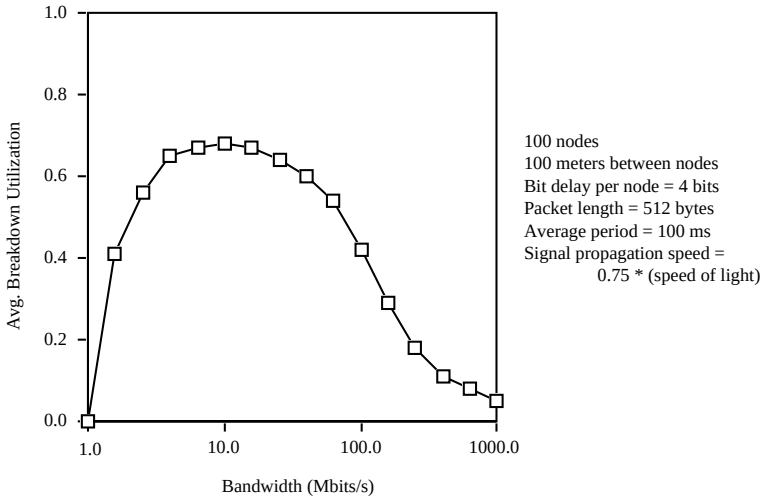

Figure 4. Average Breakdown Utilization with Rate Monotonic Scheduling using the IEEE 802.5 Protocol

4. *Asynchronous message handling.* When using the rate monotonic algorithm, asynchronous messages can be given a reasonable response time. They are transmitted at a higher priority up until the point at which the deadlines of synchronous messages would start to be missed. A fast response time for asynchronous messages will, on average, increase the probability that real-time asynchronous messages will be able to satisfy their time constraints.

5. *Runtime overhead.* In a priority driven protocol, priority arbitration takes some time to complete. This reduces the amount of time that can be spent transmitting messages. When the network bandwidth increases, the relative overhead required for priority arbitration becomes more significant. This can be clearly seen in Figure 4. The average breakdown utilization decreases rapidly when the network bandwidth approaches 100 Mbits/s. This is because when the network bandwidth increases, the physical length of packets on the ring decreases. Because the 802.5 standard allows only a single token on the ring at any time, smaller packet sizes will result in a larger portion of the ring being idle. This results in a lower breakdown utilization.

Most existing literature on rate monotonic scheduling assumes that deadlines are equal to periods. The worst case achievable utilization derived by Liu and Layland relies on this assumption. Lehoczky (1990) and Shih et al. (1993) have extended the rate monotonic analysis to allow deadlines to differ from periods. These results are for a centralized environment with negligible scheduling overhead. Nevertheless, these results could be extended to a network environment by augmenting the message lengths and allowing for blocking time as above.

The preceding discussion on the access arbitration based approach to meeting synchronous message deadlines has concentrated on rate monotonic scheduling with priority driven protocols. Another possibility is to use a centralized access arbitration process that follows a preplanned schedule in granting access to the channel. This technique is used with the MIL-STD-1553 bus. The 1553 bus is widely used in embedded real-time systems, for example, in military aircraft such as the F-15 Eagle (Ziegler 1989). In the 1553 bus, the access arbitration process is concentrated in a single bus master. All message traffic is initiated by the bus master. Nodes with a message to send wait passively until the bus master explicitly gives them permission to start sending the message. Synchronous message transmission is scheduled on a timeline at system design time. The access arbitration process on the bus master follows this schedule in giving nodes permission to send messages.

3.2. The Transmission Control Based Approach

The preceding discussion has concentrated on using the access arbitration process to guarantee that message deadlines will be met. We now shift focus and examine the use of the transmission control process in meeting message deadlines. In particular, the transmission control process of the timed token protocol is examined to determine how to allocate the channel bandwidth among the nodes so that the synchronous messages are guaranteed.

3.2.1. The Timed Token Protocol

The timed token protocol (Grow 1982) is a token passing protocol in which the amount of time that a node may hold the token is bounded. It has been incorporated into many network standards, including the Fiber Distributed Data Interface (FDDI) (ANSI 1990), the IEEE 802.4 token bus standard (ANSI 1985; Jayasumana and Jayasumana 1989; Montuschi 1992), and the Survivable Adaptable Fiber Optic Embedded Network (SAFENET) (Green and Marlow 1989; Paige 1990; U. S. Department of Defense 1994).

In the timed token protocol, nodes are arranged in a ring. When a node releases the token, the token is passed directly to the following node in the ring. Each node i in the ring is assigned a constant value, H_i, known as its *synchronous bandwidth*.[4] At network initialization time, a protocol parameter called the Target Token Rotation Time ($TTRT$) is specified. The $TTRT$ gives the expected token rotation time. The FDDI standard requires that $TTRT$ be no more than half of the minimum message deadline.[5] Upon gaining possession of the token, the transmission control process at a node will allow the node to send synchronous messages for at most H_i time units. If the token arrived earlier than expected, the transmission control process will then also permit the transmission of asynchronous messages.

Even though the token rotation time is bounded in the timed token protocol, a node may still be unable to guarantee synchronous messages. This may be the

result of the node being granted an inadequate fraction of the $TTRT$. On the other hand, allocating excess amounts of synchronous bandwidth to nodes may increase the token rotation time. This can also cause message deadlines to be missed. Guaranteeing synchronous message deadlines is thus dependent on an appropriate allocation of the synchronous bandwidth.

3.2.2. Allocating Synchronous Bandwidth

Some of the synchronous bandwidth allocation schemes that have been studied (Agrawal et al. 1992, 1993) are listed below:[6]

- The *full length allocation scheme.* In this scheme, the synchronous bandwidth H_i allocated to node is equal to the time required for transmitting a synchronous message arriving at that node:

$$H_i = C_i. \tag{8}$$

- The *equal partition allocation scheme.* In this scheme, the usable portion of the token rotation time is divided equally among the nodes. Recall that the propagation delay is τ. Thus the portion of the token rotation time that can be used for sending messages is $TTRT - \tau$. Hence the synchronous bandwidths are allocated as below:

$$H_i = \frac{TTRT - \tau}{n}. \tag{9}$$

- The *proportional allocation scheme.* In this scheme, the usable portion of the token rotation time is divided among the nodes according to the ratio of message lengths C_i to message periods T_i:

$$H_i = \frac{C_i}{T_i}(TTRT - \tau). \tag{10}$$

- The *normalized proportional allocation scheme.* In this scheme, the bandwidths used in the proportional allocation scheme are normalized by the total utilization of the synchronous messages:

$$H_i = \frac{C_i/T_i}{U_S}(TTRT - \tau), \tag{11}$$

  where $U_S = \sum_{i=1}^{n} \frac{C_i}{T_i}$.

- The *local allocation scheme.* In this scheme, the message length is divided by the worst case number of token visits to a node during a single message period. It is known that the token will visit node i at least $(\lfloor T_i/TTRT \rfloor - 1)$ times during

Table 1. Summary of the Synchronous Bandwidth Allocation Schemes.

Name	Formula for H_i	W.C.A.U.
Full length	$H_i = C_i$	0
Proportional	$H_i = \frac{C_i}{T_i} \cdot (TTRT - \tau)$	0
Equal partition	$H_i = \frac{TTRT - \tau}{n}$	$\frac{1-\alpha}{3n-(1-\alpha)}$
Normalized proportional	$H_i = \frac{C_i/T_i}{U} \cdot (TTRT - \tau)$	$\frac{1-\alpha}{3}$
Local	$H_i = \frac{C_i}{\lfloor T_i / TTRT \rfloor - 1}$	$\frac{1-\alpha}{3}$

* *W.C.A.U.* is an abbreviation of 'Worst Case Achievable Utilization.'

* $\alpha = \tau / TTRT$.

a period of length T_i (Agrawal et al. 1993). Thus the synchronous bandwidth is allocated as given below:

$$H_i = \frac{C_i}{\lfloor T_i / TTRT \rfloor - 1}. \tag{12}$$

Table 1 shows the worst case achievable utilizations of the synchronous bandwidth allocation schemes introduced above. These results were derived using the assumption that $D_i = T_i$, i.e., the deadline of a message is the end of the period in which it arrived. The full length and the proportional allocation schemes have a worst case achievable utilization of 0. The equal partition allocation scheme has a worst case achievable utilization that approaches 0 as the number of nodes in the network increases. Both the normalized proportional allocation scheme and the local allocation scheme, on the other hand, have a worst case achievable utilization of 0.33. If the utilization of a set of synchronous message streams is less than 0.33 of the usable network bandwidth, then the synchronous messages will be guaranteed by these allocation schemes. The remaining 0.67 of the usable network bandwidth can be used for the transmission of asynchronous messages.

From Table 1, it can be seen that the normalized proportional scheme and the local scheme both have the same worst case achievable utilization. An advantage of the local allocation scheme, however, is that it only uses information local to node i in calculating the synchronous bandwidth H_i. Information local to node i includes $TTRT$, τ, and the characteristics of stream S_i. The benefit of using only local information is that if the characteristics of stream S_i change, only H_i needs be recalculated. The synchronous bandwidths on other nodes can remain unchanged because they are not dependent on S_i. This contrasts with a scheme that uses nonlocal information, such as the normalized proportional allocation scheme, in which the synchronous bandwidths on all nodes would have to be recalculated.

3.2.3. Comments

As with the access arbitration based approach, we now discuss the issues of synchronous message utilization, robustness, timing faults, asynchronous message handling, and runtime overhead for the timed token protocol.

1. *Synchronous message utilization.* The timed token protocol has a worst case achievable utilization of at least 33% of the usable network bandwidth if the synchronous bandwidths are allocated appropriately. This result is achieved without incurring the cost of global priority arbitration. Similarly to rate monotonic scheduling with priority driven protocols, the average breakdown utilization is also of interest for the timed token protocol (Kamat and Zhao 1993). Figure 5 shows the average breakdown utilization for the timed token protocol using the local allocation scheme. The figure shows that for the network configuration examined, the average breakdown utilization increases with the network bandwidth. The average breakdown utilization is significantly greater than the worst case achievable utilization when the bandwidth is around 100 Mbits/s, which is a typical operating range for the timed token protocol.

2. *Robustness.* As shown in Table 1, the worst case achievable utilization of many synchronous bandwidth allocation schemes is known. The characteristics of synchronous message schemes can be altered and all messages will still meet their deadlines, provided that the utilization of the synchronous message streams remains below the worst case achievable utilization.

3. *Timing faults.* If a synchronous message stream violates its specification by reducing the interarrival period or increasing the message length, only the offending stream will suffer missing deadlines. Other synchronous streams will not suffer because the transmission control process will prevent the offending stream from using more than its share of the network bandwidth.

4. *Asynchronous message handling.* In addition to guaranteeing synchronous message streams, it is also important to maximize the throughput of asynchronous The throughput of asynchronous messages is the number of non real-time messages transmitted per unit of time. Failure to maximize the throughput can result in an increased likelihood of buffer overflow. Ciminiera et al. (1989) have derived a lower bound for the minimum amount of throughput for asynchronous messages with the timed token protocol. Shin and Zheng (1993) show how to use the timed token protocol to both guarantee a certain amount of synchronous traffic and to maximize the throughput of asynchronous messages. If asynchronous messages must have their deadlines guaranteed with the timed token protocol, then either the periodic server or the conservative estimation strategies can be used (Section 4.1).

5. *Runtime overhead.* The timed token protocol has less runtime overhead than a priority driven protocol. The access arbitration process merely passes the

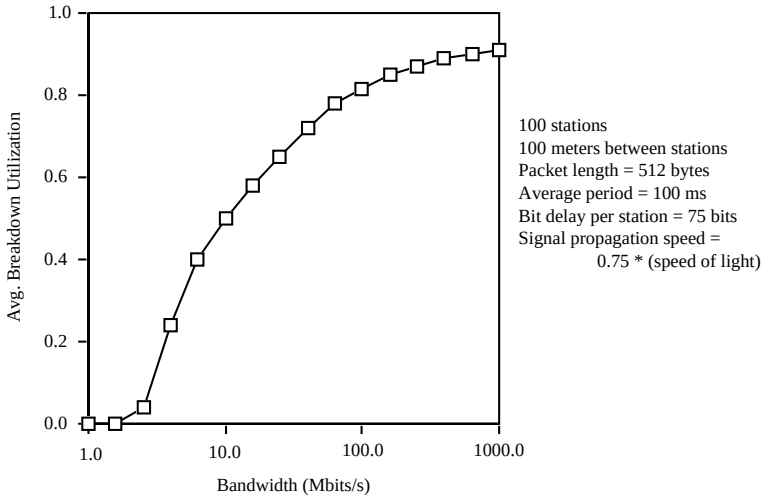

Figure 5. Average Breakdown Utilization with the Timed Token Protocol using the Local Bandwidth Allocation Scheme

token from one node to the succeeding node. If the timed token protocol is implemented on a ring, then there may be several packets on the ring at any time (as in the FDDI standard). In this case the timed token protocol can achieve reasonable utilization even when the network bandwidth is large.

The synchronous bandwidth allocation schemes discussed above are not optimal. An optimal synchronous bandwidth allocation scheme is able to guarantee that the message deadlines will be met whenever it is possible to do so. An optimal synchronous bandwidth allocation scheme has been developed (Chen et al. 1992), and is known to have a worst case achievable utilization between 33% to 40%. However, the optimal scheme is more complicated than the nonoptimal schemes discussed here, and might be difficult to implement. Given that some of the nonoptimal schemes have a worst case achievable utilization close to the optimal value, it may be preferable to use one of the nonoptimal schemes in practice. All of the work discussed in this subsection assumes that deadlines are equal to periods. Some preliminary work has been done for the case where deadlines are not equal to periods (Lim et al. 1992; Malcolm and Zhao 1993; Zheng and Shin 1993).

In addition to synchronous bandwidth allocation with the timed token protocol, another transmission control based approach to guaranteeing synchronous message deadlines is to use round robin scheduling among the synchronous message streams. For example, the TDMA protocol can be used with this technique (Kopetz and Grünsteidl 1994). In the TDMA protocol, time is divided into frames and further into slots. Synchronous messages can be guaranteed by dedicating one or more slots in each frame to messages from a given stream. A stream that is allocated the ith

slot of a frame will have exclusive access to the ith slot of every succeeding frame. The same technique can be used with Expressnet (Tobagi et al. 1983). Expressnet implements round robin scheduling on an S-shaped bus, and is especially suited for use in the high bandwidth domain. The use of TDMA or round robin scheduling for real-time messages has been discussed extensively in (Kurose et al. 1984). Due to space limitations, that material is not repeated here.

Another transmission control based approach to guaranteeing synchronous messages involves using a slotted ring network, such as FDDI-II (Teener and Gvozdanovic 1989) or Orwell (de Prycker 1991). In a slotted ring, the ring is divided into a number of fixed length slots, each capable of holding one packet (Ng and Liu 1991). By dedicating one or more slots to a given stream, or by allowing a stream to use a certain number of slots over a given time interval, synchronous messages can be guaranteed to meet their deadlines (Mukherjee et al. 1993).

4. Communication with Asynchronous Messages

Asynchronous messages arrive randomly during runtime and must therefore be scheduled dynamically. This means that using the guarantee strategy can be more complicated for asynchronous messages than for synchronous messages. Unlike the case for synchronous messages, both the access arbitration and the transmission control processes must usually be carefully considered when guaranteeing that asynchronous messages will meet their deadlines. This topic is considered further in Section 4.1.

It may not always be possible or even desirable to use the guarantee strategy for asynchronous messages. For example, there may be insufficient resources, such as buffer space or network bandwidth, to handle all contingencies with respect to asynchronous messages. Alternatively, an application may be prepared to suffer some asynchronous message loss in order to take advantage of statistical multiplexing and achieve a higher overall network utilization. In these cases, the best-effort strategy is used for asynchronous messages.

The work on best-effort scheduling of asynchronous messages has emphasized either the access arbitration or the transmission control process. The access arbitration based approach attempts to give channel access to nodes in an "optimal" fashion. Nodes whose messages have small deadlines are given precedence over nodes whose messages have large deadlines. This approach is discussed in Section 4.2. The transmission control based approach dynamically adjusts the transmission time assigned to a node according to the network traffic and/or message time constraints. If the network traffic is high or the message time constraints are tight, then a node will be given less transmission time. This approach is discussed in Section 4.3.

One should realize that scheduling hard real-time asynchronous messages is a relatively recent area of research, and much remains to be done. For example, in addition to minimizing the proportion of messages that miss their deadlines, one may wish to give precedence to messages that are more "critical" to the application.

Partly because of the ongoing nature of work in this area, no consensus has arisen as to the "best" way to handle asynchronous messages. Consequently, some of the work discussed in this section is more conceptual in nature, and has not yet been implemented in existing systems. In the following sections, each of the approaches listed above will be considered in turn.

4.1. The Guarantee Strategy with Asynchronous Messages

The guarantee strategy should be used for asynchronous messages when it is essential that the messages meet their deadlines. If the maximum generation rate of asynchronous messages is known, then messages may be guaranteed at system design time by providing sufficient network bandwidth (Ramamritham 1987).[7] Otherwise, asynchronous messages must be guaranteed soon after their arrival in the system, before the actual transmission of the message has begun. This provides an early warning if a deadline may be missed, allowing more time for any necessary recovery action.

Three basic methods can be followed by protocols that attempt to guarantee asynchronous messages:

1. *Periodic server.* With this method, a periodic server is maintained for each source of asynchronous messages that must be guaranteed.

2. *Conservative estimation.* With this method, messages are guaranteed based on the estimated amount of time that may pass before a given node can access the channel. As long as the estimate is always conservative, message guarantees will always be valid. This method requires that a bounded access time protocol be used.

3. *Dynamic reservation.* With this method, special control messages are transmitted to reserve future access to the channel.

Asynchronous messages that are used with the guarantee strategy may have a *scheduling arrival time* and a *scheduling deadline* in addition to the usual (physical) arrival time and deadline. For example, when scheduling a real-time task for execution on a processor, it may be known that the task must send some messages before completion. The task cannot be guaranteed unless its communication requirements can also be guaranteed. At scheduling arrival time, the time constraints and the length of a message are known. By the scheduling deadline, the decision must be made as to whether the message can be guaranteed.

4.1.1. The Periodic Server Method

With this method, the access arbitration process maintains a periodic server for each source of asynchronous messages that require guarantees. In other words, the

asynchronous messages are treated as synchronous messages. Each server is guaranteed to have a certain amount of channel access during its period. This access time can then be used by the transmission control process to send any pending guaranteed asynchronous messages (Strosnider et al. 1988). If the worst case behavior of asynchronous messages is known, then the periodic server approach can be used to guarantee asynchronous messages at design time. Runtime guarantees can also be issued because the protocol can determine at the scheduling arrival time of a message how much transmission time is available. The advantage of a periodic server is that it can often be implemented with little overhead. The disadvantage of a periodic server is that it reduces the channel utilization available for synchronous messages.

4.1.2. The Conservative Estimation Method

The basic idea behind the conservative estimation method is as follows. When a new asynchronous message arrives at a node, the node calculates the worst case time by which transmission of the message will be completed. If this time is less than the deadline of the message, then the message can be guaranteed.

Several points need to be noted when using this method. First, in order to calculate the worst case time by which transmission of a message will be completed, a bounded access time protocol must be used. Second, calculation of the worst case time of transmission must allow for any other messages that have been guaranteed at the node. If allowance is not made for previously guaranteed messages, a node may end up "guaranteeing" more messages than it has time to send.

To illustrate the operation of the conservative estimation method, we give an example using the TDMA protocol. Suppose there are three nodes in the network, and that node 1 will have access to the channel during time intervals $(0,1),(3,4),(6,7),\ldots$ (Figure 6). Suppose further that node 1 has already guaranteed two messages of length 1, one with a deadline of 2 and one with a deadline of 7. If a new message of length 1 arrives at node 1 at time 0 with deadline 5, then it can be scheduled for transmission during time interval (3,4). The message can thus be guaranteed at time 0. Note that the new message cannot be sent during interval (0,1), as this violates an earlier guarantee given by node 1.

Conservative estimation can be used in conjunction with any protocol with a bounded access time. For example, conservative estimation can be used with token passing protocols if the amount of time that a node can hold the token is bounded (Malcolm et al. 1990). For CSMA/CD networks, Arvind (1991) has proposed a new window protocol that has a bounded access time. This protocol is similar to the window protocols discussed earlier, but the window is managed in such a way that the maximum amount of time that a node may have to wait before sending a message is bounded.

The advantage of the conservative estimation method is that it has a relatively low overhead (no extra messages are required to be sent). However it may be too pessimistic in granting guarantees when the network is lightly loaded. For example,

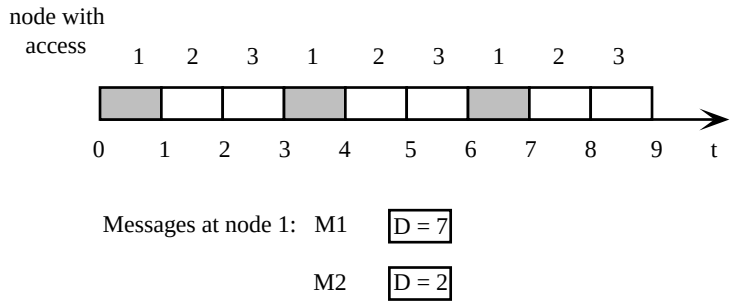

Figure 6. Conservative Estimation using the TDMA Protocol

consider if a token passing protocol is used in which each node may only transmit a single message before releasing the token. A conservative estimate by a node of the amount of time in the future in which it will be able to send messages must assume that all nodes in the system will always have a message to send. If this is not what actually happens, then the node may have more channel access time than was anticipated.

4.1.3. The Dynamic Reservation Method

The basic idea behind dynamic reservation is to dynamically reserve future access to the channel. When a new message M arrives at a node n, the node attempts to reserve a future time interval to transmit M. During this time interval, node n will be guaranteed of exclusive access to the channel, and can transmit message M. Node n attempts to reserve the interval of time by broadcasting a control message to all other nodes. The control message informs the other nodes that a given interval of time is reserved for use by node n. The control message itself has a deadline, corresponding to the scheduling deadline of the message it is attempting to guarantee. If the control message cannot be sent by its deadline, then message M is not guaranteed.

The dynamic reservation method has been adopted for use with both a CSMA/CD window protocol (Malcolm et al. 1990) and a token passing protocol (Malcolm et al. 1990). When the network load is relatively low, both protocols perform better than TDMA or token passing protocols using the conservative estimation strategy. When the network load becomes higher, the extra overhead of the control messages results in a poorer performance than is achieved with the conservative estimation method. In general, dynamic reservation would only be used if the proportion of messages that require guarantees is relatively small.

4.2. Best-Effort Scheduling: The Access Arbitration Based Approach

We now consider scheduling asynchronous messages with the best-effort strategy. The work in this area can be classified according to whether it emphasizes the access arbitration process or the transmission control process. At first, we will discuss the access arbitration based approach. The idea behind the work in this area is that the access arbitration process should grant channel access to nodes in an "optimal" fashion. Channel access is granted to nodes in such a way that the number of messages that miss their deadlines is minimized. First, we consider the appropriate choice of access arbitration policy. Second, we consider the implementation of optimal access arbitration policies in MAC protocols for multiple-access networks. Finally, we consider the effect of scheduling overhead on the performance of protocols that implement an optimal access arbitration policy.

4.2.1. Choice of Access Arbitration Policy

Scheduling hard real-time asynchronous messages in a network is similar to dynamic processor scheduling of hard real-time aperiodic tasks. For processor scheduling, the minimum-laxity-first (MLF) policy is known to be optimal for a wide range of uniprocessor and multiprocessor systems (Panwar and Towsley 1988; Panwar et al. 1988).[8] Furthermore, it is known that sometimes even a policy that only approximates the MLF policy can still result in a significant reduction in task loss (Hong et al. 1989; Zhao and Stankovic 1989). Consequently, many protocols for asynchronous messages use an access arbitration policy that gives access to nodes based on the MLF policy.

4.2.2. Implementation of the MLF Policy

Using Priority Driven Protocols

The MLF policy can be implemented by a priority driven protocol that supports global priority arbitration. Examples of such protocols include the IEEE 802.5 protocol (IEEE 1989) for ring networks and the priority driven protocol for bus networks that were discussed in Section 3.1.3. The basic idea is to assign message priorities based on message laxities; messages with a smaller laxity are assigned a higher priority. By assigning priorities appropriately and using global priority arbitration, protocols such as IEEE 802.5 can reduce the probability that messages will miss their deadlines (Shin and Hou 1990).

However, several problems inherent in this approach may impact on the network performance.

- First, priority driven protocols only approximate the MLF policy. Priority arbitration requires some time to complete. This can result in messages being lost

while priority arbitration is in progress. Alternatively, high priority messages that have just arrived may be overlooked if they arrived in the middle of priority arbitration.

- Another problem is that priorities must be assigned afresh each time that priority arbitration occurs. Pending laxities become more urgent as time progresses. Hence, the priority of a message needs to be increased if the message remains unsent. With current network implementations, however, it can be difficult to change the priority of a message once the message has been queued for transmission.
- Decreased performance can also result if the protocol supports an insufficient number of priority levels. Because there are typically many more possible values for laxities than there are for priority levels, messages with different laxities may be mapped to the same priority level. This can result in priority inversion, in which a message with a larger laxity is transmitted first. An occurrence of priority inversion can potentially cause another message to be lost (Yao and Zhao 1991). The IEEE 802.5 protocol, for example, can send only 50% of the messages that would be transmitted by an exact implementation of the MLF policy in the worst case (Yao and Zhao 1991). This result is independent of the number of priority levels and the priority assignment function. In the average case, however, priority driven protocols can perform close to the exact MLF policy when there are a sufficient number of priority levels and when a suitable priority assignment function is used (Yao 1994).

Using Virtual Time Protocols

Virtual time protocols are designed for use with CSMA/CD networks (Molle and Kleinrock 1985). One of the key issues in implementing a global MLF policy is to determine the message with the minimum laxity in the system. To achieve this, some kind of communication among the nodes must occur. In a ring network, nodes can explicitly communicate their state to other nodes by writing in the token. This approach cannot be used in a CSMA/CD network. Although nodes can broadcast their local state in a CSMA/CD network, the time required to do so dramatically reduces the time available for transmitting other messages.

However, in a CSMA/CD network, the access arbitration process at a node can use the state of the channel itself to infer information about other nodes in the network. The channel is *idle* if no nodes are attempting to transmit a message. The channel is *busy* if one node is currently transmitting a message. The channel is in a *collision* state if two or more nodes are currently transmitting messages. Information about the channel state is globally available to all nodes, and can be obtained almost instantaneously. The protocols that will now be considered use the state of the channel to implement a global MLF policy in CSMA/CD networks.

The virtual time protocols implement the MLF policy as follows. When the channel becomes idle, the access arbitration process at a node with a message to

send waits for an interval of time proportional to the laxity of the message. At the expiration of its waiting time, the node is given permission to send the message provided that the channel is still idle. If a collision occurs, then there is another node with a message of the same laxity. The access arbitration processes then attempts to resolve the collision probabilistically. Nodes involved in the collision either retransmit the message with probability p, or wait for an interval of time proportional to their laxity. The scheme outlined above is the basic approach taken by the virtual time protocol described in (Zhao and Ramamritham 1986, 1987).

The simulation data in (Zhao and Ramamritham 1986, 1987) show that the virtual time protocol can perform close to the exact MLF policy under a wide range of network load conditions. However, the proportionality constant used to determine the amount of waiting time must be chosen with care. Because of the propagation delay of the network, nodes do not instantaneously detect when another node begins transmission. If the proportionality constant is small, several nodes may begin transmission at approximately the same time, even though their messages have different laxities. This results in a channel collision. On the other hand, a large proportionality constant can result in a large amount of idle time. These factors make the virtual time protocol sensitive to the value of the constant used to determine the waiting time. Note that the virtual time protocol only approximates the MLF policy. This is because when collisions are resolved, they may be resolved in favor of a message with a larger laxity.

Using Window Protocols

The virtual time protocols were proposed for use in CSMA/CD networks. *Window protocols* have been proposed to implement the MLF policy in both CSMA/CD networks and token ring networks. Traditionally, window protocols have been used to arbitrate access to the channel in a fair manner, for example, on a first-come-first-served basis. In this kind of window protocol, nodes in the network agree on a common interval or window, (low, up), on the time axis (Kurose et al. 1984). A message is regarded as being in the window if the arrival time of the message lies in the interval (low, up). If only one node has a message in the window, then that message will be sent. If several nodes have messages in the window, the window size is reduced until only one message is in the window. Finally, if no nodes have a message in the window, then the window is shifted further along the time axis. Management of the window is the responsibility of the access arbitration process.

Several papers have addressed the modification of the traditional window protocol in order to implement the MLF policy (Kurose 1984; Kurose et al. 1988; Lim et al. 1991; Zhao et al. 1988, 1990; Znati 1991). The major step in the modification is to base the window on the latest send times of messages. The latest send time of a message is the time by which transmission of the message must begin if the message is to meet its deadline. A message is in the window if its latest send time is in the interval (low, up). To ensure that the MLF policy is followed, the bottom

of the window, *low*, is maintained as the current time t. The message that is sent will then have its latest send time closest to the current time.

An advantage of window protocols as opposed to priority driven protocols is that window protocols are not limited by the number of available priority levels. Hence, they can reduce priority inversion. Window protocols are also often relatively insensitive to the initial window size chosen by the protocol (Zhao et al. 1988, 1990). A disadvantage of window protocols is that the time and space required to maintain the window can incur substantial overhead.

The key issue in designing a window protocol is to enable all nodes to maintain a consistent view of the window information. The *window information* includes the bounds of the window and the number of nodes with messages in the window. With this information, nodes can operate and adjust the window correctly. The window information is not necessarily available locally at a node. For example, information regarding the number of nodes with messages in the window is distributed around the network. The difficulty in designing a window protocol lies in ensuring that all nodes have a consistent view of the window information so that the window is properly operated and adjusted. We now discuss implementation techniques for window protocols, focusing on how to keep the window information consistent.

Window protocols in a CSMA/CD network. In a CSMA/CD network, each node stores a local copy of the window. The channel state is then used to determine the number of nodes with messages in the window. When the channel is idle, nodes with a message whose latest send time is in the window will transmit the message. There are then three possibilities, depending on whether there are zero, one, or more nodes with a message in the window:

- If there are no nodes with a message in the window, then the channel will remain idle.
- If there is only one node with a message in the window, then that message will be transmitted and the channel will be busy.
- If at least two nodes have a message in the window, then a channel collision will occur.

If the channel remained idle, then all nodes will increase the window size. If a collision occurred, then all nodes will decrease the window size. The lower portion of the original window on the time axis will be chosen as the new window. In this way, when a collision is resolved, the message that is transmitted will have the minimum laxity in the system. All nodes will have a consistent view of the window information, because each node has a local copy of the window, and each node knows the channel state. Thus the window information on each node will be maintained consistently.

Kurose et al. (Kurose 1984; Kurose et al. 1988) first proposed a window protocol for real-time communication in CSMA/CD networks. In this protocol, all message are assumed to have the same (constant) laxity when they arrive in the system. This assumption can be relaxed to allow messages to have arbitrary laxities (Zhao et al.

1988, 1990). Further improvements have been proposed for the window protocol, resulting in faster collision resolution and implementation of the exact MLF policy (Znati 1991).

Window protocols in token ring networks. In a token ring network, the upper and lower bounds of the window are stored in the token (Lim et al. 1991). The token also has a counter field that records the number of nodes with messages in the window. When the token circulates around the ring, nodes with a message in the window will increment the counter. When the token arrives back at the original sending node, the counter contains the number of messages in the window. The sending node can then directly determine the window information, and update the window accordingly:

- If there are no nodes with a message in the window, then the counter will be 0. In this case the window is moved further along the time axis.
- If there is only one node with a message in the window, then the counter will be 1. In this case the token will be passed to the node that has a message in the window, and the message will be sent.
- If at least two nodes have a message in the window, then the counter will be greater than 1. In this case the window size will be reduced, the counter set to 0, and the token passed around the ring again.

Implementing the token window protocol directly would require a lengthy token. However, a coding scheme has been designed that allows an eight-bit token to be used while still preserving the full functionality of the protocol (Yao 1994).

4.2.3. Effect of Scheduling Overhead

Although the theory of hard real-time scheduling shows that the MLF policy is optimal in the static and the dynamic cases, these results are obtained for an environment where there is no scheduling overhead. In contrast, the scheduling overhead of network protocols can be quite high. Consequently, it is possible that a protocol with a large overhead that implements the "optimal" MLF policy may have a poorer performance than a simpler but "suboptimal" protocol. The time taken to determine the message with the minimum laxity may be significant enough to result in message loss. In order to investigate these effects, the performance of three different token ring protocols has been analytically compared (Lim et al. 1991).[9] The protocols compared were the token passing protocol,[10] a priority driven protocol, and the window protocol.

Each of these protocols has a different amount of overhead. The token passing protocol has the lowest overhead, but does not follow the MLF policy. The priority driven protocol approximates the MLF policy by mapping message deadlines to appropriate priority levels. The window protocol avoids the priority inversion inherent in the priority driven protocol, but has a higher overhead.

The results obtained in the study are summarized in Figure 7. The horizontal axis corresponds to the number of nodes in the network. The vertical axis gives the node-to-node delay of the network. All messages are assumed to be of unit length, and so the node-to-node delay is normalized in terms of the message length. The priority driven protocol is assumed to have enough priority levels to uniquely represent each of the message laxities. The lines plotted on the graph divide the regions in which each of the protocols performs best in terms of minimizing the message loss. As can be seen, none of the protocols always outperforms the others. When the number of nodes is large, the token passing protocol performs best because it has the least amount of scheduling overhead. When there is a small number of nodes, the priority driven protocol performs best because the cost incurred by priority arbitration is low and the number of priority levels is sufficient. When there is a moderate number of nodes, the window protocol performs best provided that the node-to-node delay is small. When the node-to-node delay increases, the overhead incurred by the window protocol outweighs the benefits provided by the MLF policy.

The results in Figure 7 highlight the distributed nature of network scheduling. Algorithms that are optimal for centralized scheduling may not always perform well in a distributed environment. In any algorithm for network scheduling, it is important to consider the effect of the overhead of the algorithm on the final performance.

4.3. Best-Effort Scheduling: The Transmission Control Based Approach

In Section 4.2, we considered best-effort scheduling in which the access arbitration process is emphasized. We now shift attention to the transmission control process, and consider how the transmission control process can be used to maximize the proportion of messages that satisfy their time constraints. By definition, the transmission control process is used to determine how long a *node* may continue to send messages. This concept can be extended further to allow the transmission control process to determine how long each *message* may be sent.

The motivation behind allowing the transmission control process to decide how long to send a message is that in many situations, a tradeoff can be made between information completeness and message loss. In such a system, a message may have several versions. Different versions take a different amount of time to be transmitted and contain a different amount of information. When the network traffic is high and/or a message deadline is tight, it is advantageous to send a short version of the message in order to better meet message deadlines.

Many applications of multiversion messages have been developed. For example, Bially et al. (1980) have suggested such a technique for the transmission of voice traffic. Each message consists of several packets, generated from the same voice period but at different sampling rates. Messages are structured such that one or more packets can be discarded without affecting the continuity of the speech output, however, the voice quality at the receiver degrades as more packets are discarded.

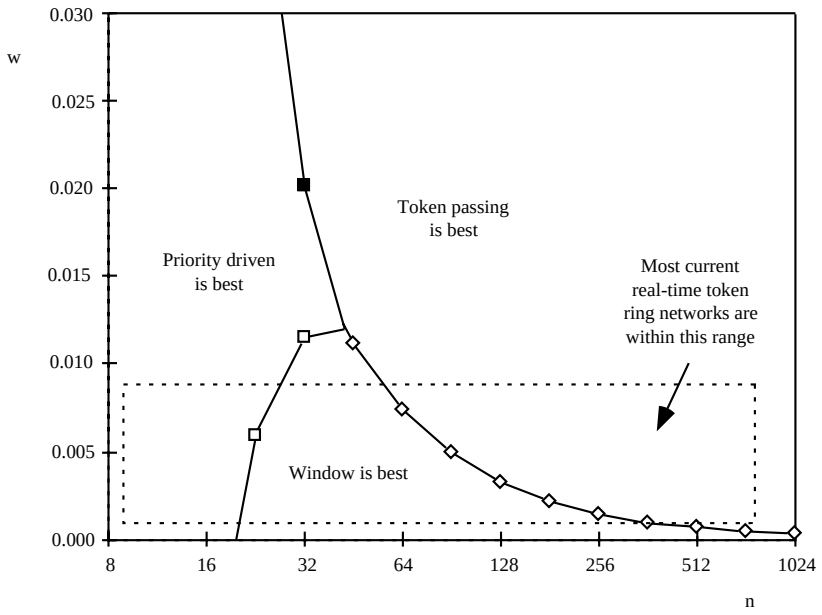

Figure 7. A Comparison of Three Token Ring Protocols

Yamamoto (1989) has discussed a layered coding scheme for video signals in ATM networks in which the higher frequency components of the video signal are placed in separate packets. There will only be a small degradation in picture quality if these packets are lost because the human eye is relatively insensitive to the higher frequency components of a video signal. Other applications of multiversion messages are discussed in (Ghanbari 1989; Karlsson and Vetterli 1989; Kim and Towsley 1986; Vetterli and Garrett 1990; Yemini 1983).

4.3.1. Issues in Version Selection for Multiversion Messages

When using multiversion messages, the transmission control process employs a *version selection scheme* to decide which version of a message should be sent. The version selection scheme is crucial to the performance of a protocol for multiversion messages. Two important points must be considered in designing a version selection scheme:

- *Timing correctness.* A version selection scheme should make the correct decision. A wrong decision may either increase the message loss (i.e., mistaken transmission of the long version of a message may result in the loss of other messages that are waiting to be sent), or may unnecessarily decrease the message information completeness.

- *Overhead.* A version selection scheme must be efficient. The version selection scheme is invoked every time that a message is sent. Thus a version selection scheme that takes a long time to make a decision would add significantly to the overhead of a protocol.

Using multiversion messages to reduce the number of missing deadlines is similar to imprecise computation (Liu et al. 1991). In imprecise computation, tasks may be terminated by the scheduler before their completion if this is necessary in order to meet the task deadlines. However, tasks that are terminated before completion will deliver poorer quality results. The scheduler thus faces a tradeoff between result quality and computation time.

In spite of this, it is difficult to directly apply the work on imprecise computation to multiversion messages. In most work on imprecise computation, the scheduler is centralized. Task characteristics are known at scheduling time. This is in contrast to the situation with multiversion messages. Network protocols are distributed, and a node usually has no knowledge of the characteristics of messages on other nodes in the network. Further, it is difficult for a node to obtain information about messages on other nodes in the network. This difficulty arises from communication delays and from the extra overhead incurred when sending extra messages to communicate state information.

Hence, most existing studies on version selection use a different approach from that used in imprecise computation. The majority of version selection schemes reported so far make the selection decision based on some estimate of the "real-time load" on a network. Unfortunately, the notion of "real-time load" has not been well defined. Do two networks with the same number of pending messages have the same real-time load if the deadlines in one of the networks are much larger? Intuitively, however, a version selection scheme should choose a shorter version of a message when doing so is the only way to meet the time constraints of the message, or when there is a lot of message traffic in the network.

In order to estimate the real-time load, version selection schemes need a measure to characterize the real-time load on the network. Version selection schemes differ in how they measure the real-time load. Some schemes use a measure based on the network behavior immediately prior to sending a message, while other schemes base their measure on the average network behavior over a recent time interval.

4.3.2. Version Selection based on Instantaneous Information

Version selection schemes based on instantaneous information base their measure of real-time load on the network behavior immediately prior to sending a message. One proposed measure of the real-time load is the number of messages queued at the sending node. For example, Kim and Towsley (1986) have proposed such a version selection scheme for packet-switching multiplexers serving real-time multipacket messages. If the packet buffer at a multiplexer contains too many packets, then the version selection scheme only selects a portion of the packets of the message to

be transmitted; other packets of the message are discarded. This version selection scheme can be incorporated easily into a network MAC protocol: if the number of messages queued at the sending node exceeds a threshold, then a shorter version of the message will be sent.

Another way to estimate the real-time load is to use the deadlines of messages (Malcolm and Zhao 1991). A shorter version of a message should be sent if there is insufficient time remaining to send the long version of the message. Though only using information local to the sending node, this approach is responsive to the network load. If the load increases, then the average message delay increases and so messages are more likely to have a shorter version sent.

4.3.3. Version Selection based on Recent History

Version selection schemes in this category base their measure of real-time load on the network behavior over a recent time interval. One such measure is the number of messages that missed their deadline at the sending node over the preceding W time units (W is a system parameter) (Zhao et al. 1989). This scheme can be used with any network topology. Another approach, that could be used in a CSMA/CD network, is to base the decision on the number of collisions that occurred in the preceding W time units (Zhao et al. 1989). Both of these approaches are effective at trading off message loss for information completeness. However, because they are based on average values, they may be unresponsive when message arrivals are bursty.

Another measure for the real-time load is the *message density*. The message density is given by the number of messages whose latest send time is in a given interval of time, normalized by the length of the time interval. Formally, the message density is defined as

$$D_T = \frac{|\{M | LS_M < T\}|}{T - t} \tag{13}$$

where t is the current time, LS_M is the latest send time of message M, and T is a parameter selected by the user. Large values of the message density mean that the amount of network traffic is high.

The message density D_T can be estimated when using a window protocol (Malcolm and Zhao 1991). The portion of the time axis under consideration by the window protocol is used in calculating the message density. In this way the density only relates to those messages with relatively urgent deadlines that are already under consideration by the window protocol. Messages with large deadlines that could be sent later will not affect decisions regarding which versions of messages with tight deadlines should be sent. The number of messages in the window, which is estimated during the normal operation of a window protocol, is used to estimate the message density over the time interval (t, *up*). This scheme is effective at reducing the proportion of messages that miss their deadlines while maintain-

Table 2. Approaches for Handling Real-Time Message Traffic

	Guarantee strategy		Best-effort strategy	
	Access arbitration based	Transmission control based	Access arbitration based	Transmission control based
Synchronous messages	Section 3.1	Section 3.2	—	—
Asynchronous messages	Section 4.1		Section 4.2	Section 4.3

ing a reasonable quality of service (information completeness), particularly when message laxities are larger (Malcolm and Zhao 1991).

5. Final Remarks

This paper has surveyed real-time communication in local area multiple-access networks. There are two general strategies that can be used in handling real-time message traffic: the guarantee strategy and the best-effort strategy. Within these strategies, there are many variations, depending on whether synchronous or asynchronous messages are considered, and whether the emphasis lies on the access arbitration process or on the transmission control process. Table 2 summarizes the different strategies and approaches to handling real-time message traffic that were discussed in this paper.

In the remainder of this section we list several other relevant issues. This list is not meant to be exhaustive, and is only intended to give the reader a flavor for related areas of research.

Clock Synchronization

Some of the protocols discussed in this paper, including the virtual time protocols for CSMA/CD networks, require that clocks on different nodes be synchronized. Further, distributed real-time systems, in general, require clock synchronization in order to enforce time constraints. Clock synchronization in distributed systems is an interesting and challenging problem. Much work has been done on bounding the possible deviation between clocks on different nodes (Dolev et al. 1986; Kopetz and Ochsenreiter 1987; Lamport and Melliar-Smith 1985; Lundelius and Lynch 1984; Ramanathan et al. 1990a, 1990b; Volz et al. 1991; Welch and Lynch 1988). However, further work remains to be done in areas such as examining the performance of probabilistic clock synchronization algorithms (Arvind 1989; Arvind et al. 1991b; Cristian 1989).

Multi-Hop Networks

The networks discussed in this paper have a simple bus or ring topology. Other networks, such as wide area networks, may have an arbitrary topology. Allowing an arbitrary topology creates many challenges. For example, in sending a real-time message between two nodes, a route must be chosen that allows the message to meet its deadline (Ferrari and Verma 1990; Leung et al. 1990). Another important area of research is the establishment of real-time channels in multi-hop networks (Ferrari and Verma 1990; Hwang et al. 1992; Kalmanek et al. 1990; Kandlur et al. 1991; Trajković and Golestani 1992; Yates et al. 1993; Zhang and Ferrari 1993; Zhang and Keshav 1991; Zhang 1990; Zheng and Shin 1992). Real-time channels are virtual circuits that are designed to handle real-time messages. However, providing performance guarantees for real-time channels can be difficult because the multiplexing of different real-time channels over the links in a wide area network distorts the characteristics (e.g., interarrival times) of the message streams. This makes it difficult to accurately model the behavior of the message traffic inside the network.

Fault Tolerance

This paper has not considered the possibility of messages becoming corrupted during transmission, or of network faults or failures. In practice, work is needed in these areas because such occurrences can cause the time constraints of messages to be violated. The two main approaches used in achieving fault tolerance are *temporal redundancy* and *spatial redundancy* (Arvind et al. 1991b; Pradhan 1986; Wilkov 1972; Zheng and Shin 1992). With temporal redundancy, errors are handled by retransmitting a message if a failure occurs. For this approach to be useful, message deadlines must be sufficiently large. Further, care must be taken to ensure that retransmission of a message does not violate guarantees for other messages. Spatial redundancy involves transmitting several copies of the message on physically distinct communication channels. For example, Kamat et al. (1994) have designed a spatially redundant network using multiple FDDI token rings. By sending multiple copies of a message over separate rings, the probability that the message will be successfully received is improved. The network also has the ability to reconfigure itself in the presence of faults so that the available bandwidth is maximized. Zheng and Shin (1992) have designed algorithms for fault tolerant real-time channels in multi-hop networks. If a fault occurs on the path between the source and destination nodes of a real-time channel, then the messages in the channel can be rerouted via another path.

Hard Real-Time Communication at the Logical Link Layer

This paper has concentrated on hard real-time communication at the MAC layer, because it is the MAC protocols that are directly responsible for scheduling and transmitting messages. However, an interface is needed between the MAC layer and higher level layers in order to take advantage of the features provided by MAC protocols. Providing this interface is the function of the logical link layer.

Arvind et al. (1991b) discuss the use of the logical link layer to provide support for real-time applications. They propose that the logical link layer provide both real-time connection oriented service and real-time connectionless service. Connection oriented service is used to provide a virtual channel for the guaranteed transmission of a stream of messages (typically synchronous messages). Connectionless service does not involve setting up a connection or guaranteeing messages, but it will attempt to satisfy the time constraints of a message. Arvind et al. (1991a, 1991b) also discuss a class of window protocols that are capable of providing both connection oriented and connectionless service.

Other Networks and Protocols

In this paper we focused on MAC protocols for local area networks. Several other networks and protocols that need to be investigated are listed below.

Spring Net. SpringNet (Stankovic et al. 1991a, 1991b) is an architecture for a network of multiprocessors, each running the Spring Kernel (Niehaus et al. 1992; Stankovic and Ramamritham 1989). Each multiprocessor is composed of application processors, system processors, and an I/O subsystem. Nodes in SpringNet are arranged in an n dimensional grid, and are connected by a series of fiber optic rings. For example, a 2-dimensional grid would have one ring for each row and one ring for each column. Global memory can be provided by maintaining replicated copies of the memory on all nodes. This global memory can be accessed with a very low latency cost, and provides a basis for predictable real-time communication.

DQDB. The work on real-time communications in local area networks needs to be extended to metropolitan area networks using protocols such as the IEEE 802.6 DQDB protocol. Because of the wide geographical expanse of metropolitan area networks, scheduling decisions regarding which messages to send must be made concurrently with delayed or incomplete information by different nodes in the network. Sha et al. (1992) have examined the scheduling of synchronous messages in dual bus networks, and have shown that under certain conditions the rate monotonic scheduling algorithm can be implemented. Further work is needed to examine the use of dual bus networks for transmitting asynchronous messages, particularly with regard to multiversion messages or message guarantees.

High-speed Networks. The speed of future networks will be of the order of 1 gigabit per second (Gbits/s) or higher. Gigabit networks introduce additional problems that must be considered. Message traffic on gigabit networks is expected to be highly bursty because of the high rate at which messages can be transmitted (Kung

1992). Due to the mismatch in bandwidth, care must be taken when traffic is routed from a gigabit network to a slower network such as Ethernet. Further, on large gigabit networks, the network latency can play a larger role than the network bandwidth in determining the response time (Kleinrock 1992). All of these factors provide additional challenges to the designers of network protocols, for example, in the area of high speed local area networks (Bhargava et al. 1988; Chlamtac and Ganz 1988; Scholl and Coden 1988; Tobagi et al. 1983; Zafirovic-Vukotic et al. 1988).

Acknowledgements

This paper surveys recent results in the field of real-time communications. Furthermore, many individuals have directly contributed to the survey itself. In particular, we would like to thank Prof. John Stankovic of the University of Massachusetts at Amherst for his encouragement. The comments from the anonymous referees improved the paper greatly. Sanjay Kamat and Swee Hor Teh of Texas A&M University also made many helpful comments regarding an earlier version of the paper. We would also like to acknowledge support from AFOSR (grant F49620-92-J-0385), NSF (grant NCR-9210583), ONR (grant N00014-92-J-4031), and Texas A&M University under an Engineering Excellence Grant.

Notes

1. This paper does not give a comprehensive introduction to multiple-access networks or MAC protocols, but there are many excellent references available for readers who wish to learn more (Schwartz 1987; Stallings 1991; Tanenbaum 1988).
2. The literature on real-time processor scheduling often refers to periodic and aperiodic tasks. The terms "periodic" and "aperiodic" are synonymous with "synchronous" and "asynchronous," respectively.
3. Note, however, that using a token ring protocol to provide priority arbitration in a bus network would be very inefficient, because of the overhead due to passing the token.
4. Some other synonymous terms that researchers use are *bandwidth allocation* (Ulm 1982), *synchronous allocation* (Jain 1991), *synchronous bandwidth assignments* (Johnson 1987), *high priority token holding time* (Pang and Tobagi 1989), and *synchronous capacity* (Agrawal et al. 1992, 1993). We use the term *synchronous bandwidth* in order to be consistent with the current FDDI standard.
5. $TTRT$ can be no more than half of the minimum message deadline because the maximum time between consecutive token visits to a node can approach $2TTRT$ (Agrawal et al. 1992; Chen et al. 1992; Johnson 1987).
6. The synchronous bandwidth allocation schemes listed below assume that there is only one stream of synchronous messages originating at each node. This assumption involves no loss of generality, because a network in which several streams may originate at each node can be transformed into a logically equivalent network in which only a single stream originates at each node (Agrawal et al. 1992).
7. If the system has advance knowledge about asynchronous messages, then the term "sporadic" is sometimes used in the literature instead of "asynchronous."

8. Another optimal policy is the earliest-deadline-first (EDF) policy. In this paper we concentrate on the MLF policy. However, the protocols discussed can easily be modified to use EDF instead of MLF scheduling.
9. This paper uses the EDF policy, but since it assumes that all messages have the same length, this is equivalent to the MLF policy.
10. This is the traditional protocol used in token ring networks. A token is passed from one node to another around the ring. A node wishing to send a message must first seize the token, and only then transmit the message. The token is then released, making it available to the next node in the ring.

References

Agrawal, G., B. Chen, W. Zhao, and S. Davari. 1992. Guaranteeing synchronous message deadlines in high speed token ring networks with timed token protocol. In *Proceedings of the 12th IEEE International Conference on Distributed Computing Systems*, pages 468–475.

Agrawal, G., B. Chen, and W. Zhao. 1993. Local synchronous capacity allocation schemes for guaranteeing message deadlines with the timed token protocol. In *Proceedings of INFOCOM '93*, pages 186–193.

ANSI Standard X3T9.5/88-139, Rev 4.0. 1990. *FDDI Media Access Control (MAC).*

ANSI/IEEE Standard 802.4-1985. 1985. *Token-Passing Bus Access Method and Physical Layer Specifications.*

Arvind, K. 1989. A new probabilistic algorithm for clock synchronization. Technical Report 88-81, Department of Computer and Information Science, University of Massachusetts at Amherst.

Arvind, K., K. Ramamritham, and J. A. Stankovic. 1991a. Window MAC protocols for real-time communication services. Technical Report 90-127, Department of Computer and Information Science, University of Massachusetts at Amherst.

Arvind, K., K. Ramamritham, and J. A. Stankovic. 1991b. A local area network architecture for communication in distributed real-time systems. *Journal of Real-Time Systems*, 3(2):115–147.

Arvind, K. 1991. *Protocols for Distributed Real-Time Systems.* PhD thesis, Department of Computer and Information Science, University of Massachusetts at Amherst.

Bhargava, A., J. F. Kurose, and D. Towsley. 1988. A hybrid media access protocol for high-speed ring networks. *IEEE Transactions on Selected Areas in Communications*, 6(6):924–933.

Bially, T., B. Gold, and S. Seneff. 1980. A technique for adaptive voice flow control in integrated packet networks. *IEEE Transactions on Communications*, COM-28(3):325–333.

Chen, B., G. Agrawal, and W. Zhao. 1992. Optimal synchronous capacity allocation for hard real-time communications with the timed token protocol. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 198–207.

Chlamtac, I., and A. Ganz. 1988. A multibus train communication (AMTRAC) architecture for high-speed fiber optic networks. *IEEE Transactions on Selected Areas in Communications*, 6(6):903–912.

Ciminiera, L., P. Montuschi, and A. Valenzano. 1989. Some properties of double-ring networks with real-time constraints. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 360–368.

Clark, D. D., S. Shenker, and L. Zhang. 1992. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of ACM SIGCOMM'92*, pages 14–26.

Cristian, F. 1989. Probabilistic clock synchronization. *Distributed Computing*, 3:146–158.

de Prycker, M. 1991. *Asynchronous Transfer Mode: Solution for Broadband ISDN.* Ellis Horwood.

Dolev, D., J. Y. Halpern, and H. R. Strong. 1986. On the possibility and impossibility of achieving clock synchronization. *Journal of Computer and System Sciences*, 32:230–250.

Ferrari, D., and D. C. Verma. 1990. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, SAC-8(3):368–379.

Ghanbari, M. 1989. Two-layer coding of video signals for VBR networks. *IEEE Journal on Selected Areas in Communications*, 7(5):771–781.

Green, D. T., and D. T. Marlow. 1989. SAFENET — a LAN for navy mission critical systems. In *Proceedings of the IEEE Conference on Local Computer Networks*, pages 340–346.

Grow, R. M. 1982. A timed token protocol for local area networks. In *Proceedings of Electro/82, Token Access Protocols*, paper 17/3.

Hong, J., X. Tan, and D. Towsley. 1989. A performance analysis of minimum laxity and earliest deadline scheduling in a real-time system. *IEEE Transactions on Computers*, 38(12):1736–1744.

Hwang, R.-H., J. F. Kurose, and D. Towsley. 1992. The effect of processing delay and QOS requirements in high speed networks. In *Proceedings of IEEE Infocom'92*, pages 160–169.

IEEE Standard 802.5-1989. 1989. *Token Ring Access Method and Physical Layer Specifications.*

Jain, R. 1991. Performance analysis of FDDI token ring networks: Effect of parameters and guidelines for setting TTRT. *IEEE LTS*, 2(2):16–22.

Jayasumana, A. P., and G. G. Jayasumana. 1989. On the use of the IEEE 802.4 token bus in distributed real-time control systems. *IEEE Transactions on Industrial Electronics*, 36(3):391–397.

Johnson, M. J. 1987. Proof that timing requirements of the FDDI token ring protocols are satisfied. *IEEE Transactions on Communications*, COM-35(6):620–625.

Kalmanek, C. R., H. Kanakia, and S. Keshav. 1990. Rate controlled servers for very high-speed networks. In *Proceedings of IEEE Global Telecommunications Conference*, pages 300.3.1–300.3.9.

Kamat, S., and W. Zhao. 1993. Real-time schedulability of two token ring protocols. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, pages 347–354.

Kamat, S., G. Agrawal, and W. Zhao. 1994. On available bandwidth in FDDI-based reconfigurable networks. To appear in *Proceedings of IEEE Infocom'94*.

Kandlur, D. D., K. G. Shin, and D. Ferrari. 1991. Real-time communication in multi-hop networks. In *Proceedings of the 11th International Conference on Distributed Computing Systems*, pages 300–307.

Karlsson, G., and M. Vetterli. 1989. Packet video and its integration into the network architecture. *IEEE Journal on Selected Areas in Communications*, 7(5):739–751.

Kim, B. G., and D. Towsley. 1986. Dynamic flow control protocols for packet-switching multiplexers serving real-time multipacket messagers. *IEEE Transactions on Communications*, COM-34(4):348–356.

Klein, M. H., J. P. Lehoczky, and R. Rajkumar. 1994. Rate-Monotonic Analysis for Real-Time Industrial Computing. *IEEE Computer*, 27(1):24–33.

Kleinrock, L. 1985. Distributed systems. *Communications of the ACM*, 28(11):1200–1213.

Kleinrock, L. 1992. The latency/bandwidth tradeoff in gigabit networks. *IEEE Communications Magazine*, 30(4):36–40.

Kopetz, H., and G. Grünsteidl. 1994. TTP — A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer*, 27(1):14–23.

Kopetz, H., and W. Ochsenreiter. 1987. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers*, 36(8):933–940.

Krishna, C. M., and Y. H. Lee, editors. 1991. Special issue on real-time systems. *IEEE Computer*, 24(5).

Kung, H. T. 1992. Gigabit local area networks: A systems perspective. *IEEE Communications Magazine*, pages 79–89.

Kurose, J. F., M. Schwartz, and Y. Yemini. 1984. Multiple-access protocols and time constrained communication. *Computing Surveys*, 16(1):43–70.

Kurose, J. F., M. Schwartz, and Y. Yemini. 1988. Controlling window protocols for time-constrained communication in multiple access networks. *IEEE Transactions on Communications*, 36(1):41–49.

Kurose, J. F. 1984. *Time-Constrained Communication in Multiple Access Networks.* PhD thesis, Department of Computer Science, Columbia University.

Lamport, L., and P. M. Melliar-Smith. 1985. Synchronizing clocks in the presence of faults. *Journal of the Association for Computing Machinery*, 32(1):52–78.

Lehoczky, J. P. 1990. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 201–209.

Lehoczky, J. P., and L. Sha. 1986. Performance of real-time bus scheduling algorithms. *ACM Performance Evaluation Review*, 14(1):44–53.

Lehoczky, J. P., L. Sha, and Y. Ding. 1989. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 166–171.

Leung, J. Y.-T., T. W. Tam, and G. H. Young. 1990. On-line routing of real-time messages. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 126–135.

Lim, C.-C., L.-J. Yao, and W. Zhao. 1991. A comparative study of three token ring protocols for real-time communications. In *Proceedings of the 11th IEEE International Conference on Distributed Computing Systems*, pages 308–317.

Lim, C.-C., L.-J. Yao, and W. Zhao. 1992. Transmitting time-dependent multimedia data in FDDI networks. In *Proceedings SPIE International Symposium, OE/FIBERS'92*, pages 144–153.

Liu, C. L., and J. W. Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61.

Liu, J. W. S., K.-J. Lin, W.-K. Shih, A. C.-S. Yu, J.-Y. Chung, and W. Zhao. 1991. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5):58–68.

Liu, M. T. 1978. Distributed loop computer networks. *Advances in Computers*, 17:163–221.

Lundelius, J., and N. Lynch. 1984. An upper and lower bound for clock synchronization. *Information and Control*, 62(2/3):190–204.

Malcolm, N., and W. Zhao. 1991. Version selection schemes for hard real-time communications. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 12–21.

Malcolm, N., and W. Zhao. 1993. Guaranteeing synchronous messages with arbitrary deadline constraints in an FDDI network. In *Proceedings of the IEEE Conference on Local Computer Networks*, pages 186–195.

Malcolm, N., and W. Zhao. 1994. *Hard real-time communication in local area networks.* To be published by Kluwer Academic Publishers.

Malcolm, N., W. Zhao, and C. Barter. 1990. Guarantee protocols for communication in distributed real-time systems. In *Proceedings of IEEE Infocom'90*, pages 1078–1086.

Mok, A. K., and S. A. Ward. 1979. Distributed broadcast channel access. *Computer Networks*, 3(5):327–335.

Molle, M. L., and L. Kleinrock. 1985. Virtual time CSMA: Why two clocks are better than one. *IEEE Transactions on Communications*, COM-33(9):919–933.

Montuschi, P., L. Ciminiera, and A. Valenzano. 1992. Time characteristics of IEEE 802.4 token bus protocol. *IEE Proceedings-E*, 139(1):81–87.

Mukherjee, S., D. Saha, M. C. Saksena, and S. K. Tripathi. 1993. A bandwidth allocation scheme for time constrained message transmission on a slotted ring LAN. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 44–53.

Ng, J. K. Y., and J. W. S. Liu. 1991. Performance of local area network protocols for hard real-time applications. In *Proceedings of the 11th International Conference on Distributed Computing Systems*, pages 318–326.

Niehaus, D., E. M. Nahum, J. A. Stankovic, and K. Ramamritham. 1992. Architecture and OS support for predictable real-time systems. Personal communication.

Paige, J. L. 1990. SAFENET — a navy approach to computer networking. In *Proceedings of the IEEE Conference on Local Computer Networks*, pages 268–273.

Pang, J., and F. A. Tobagi. 1989. Throughput analysis of a timer controlled token passing protocol under heavy load. *IEEE Transactions on Communications*, 37(7):694–702.

Panwar, S. S., and D. Towsley. 1988. On the optimality of the STE rule for multiple server queues that serve customers with deadlines. Technical Report 88-81, Department of Computer and Information Science, University of Massachusetts at Amherst.

Panwar, S. S., D. Towsley, and J. K. Wolf. 1988. Optimal scheduling policies for a class of queues with customer deadlines to the beginning of service. *Journal of the Association for Computing Machinery*, 35(4):832–844.

Pleinevaux, P. 1992. An improved hard real-time scheduling for the IEEE 802.5. *The Journal of Real-Time Systems*, 4(2):99–112.

Pradhan, D. K., editor. 1986. *Fault-Tolerant Computing: Theory and Techniques.* Prentice-Hall, volume 1.

Ramamritham, K. 1987. Channel characteristics in local-area hard real-time systems. *Computer Networks and ISDN Systems*, 13(1):3–13.

Ramanathan, P., D. D. Kandlur, and K. G. Shin. 1990a. Hardware-assisted software clock synchronization for homogeneous distributed systems. *IEEE Transactions on Computers*, 39(4):514–524.

Ramanathan, P., K. G. Shin, and R. W. Butler. 1990b. Fault-tolerant clock synchronization in distributed systems. *IEEE Computer*, 23(10):33–42.

Scholl, F. W., and M. H. Coden. 1988. Passive optical star systems for fiber optic local area networks. *IEEE Transactions on Selected Areas in Communications*, 6(6):913–923.

Schulzrinne, H., J. F. Kurose, and D. Towsley. 1990. Congestion control for real-time traffic in high-speed networks. In *Proceedings of IEEE Infocom'90*, pages 543–550.

Schwartz, M. 1987. *Telecommunication Networks: Protocols, Modeling and Analysis.* Addison-Wesley.

Sha, L., R. Rajkumar, and J. Lehoczky. 1990. Real-time scheduling support in Futurebus+. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 331–340.

Sha, L, editor. 1994. Special issue on industrial computing. *IEEE Computer*, 27(1).

Sha, L., S. S. Sathaye, and J. K. Strosnider. 1992. Scheduling real-time communication on dual-link networks. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 188–197.

Shih, W. K., J. W. S. Liu, and C. L. Liu. 1993. Modified rate-monotonic algorithm for scheduling periodic jobs with deferred deadlines. *IEEE Transactions on Software Engineering*, 19(12):1171–1179.

Shin, K. G, editor. 1987. Special issue on real-time systems. *IEEE Transactions on Computers*, 36(8).

Shin, K. G., and C.-J. Hou. 1990. Analysis of three contention protocols in distributed real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 136–145.

Shin, K. G., and Q. Zheng. 1993. Mixed time-constrained and non-time-constrained communications in local area networks. *IEEE Transactions on Communications*, 41(11):1668–1676.

Stallings, W. 1991. *Data and Computer Communications.* Macmillan, 3rd edition.

Stankovic, J. A. 1984. A perspective on distributed computer systems. *IEEE Transactions on Computers*, C-33(12):1102–1115.

Stankovic, J. A. 1988. Misconceptions about real-time computing: A serious problem for next generation systems. *IEEE Computer*, 21(10):10–19.

Stankovic, J. A., D. Niehaus, and K. Ramamritham. 1991a. SpringNet: A scalable architecture for high performance, predictable, and distributed real-time computing. Technical Report 91–74, Department of Computer and Information Science, University of Massachusetts at Amherst.

Stankovic, J. A., D. Niehaus, and K. Ramamritham. 1991b. SpringNet: A scalable architecture for high performance, predictable, and distributed real-time computing. In *Workshop on Architecture Supports for Real-Time Systems*, pages 11–16.

Stankovic, J. A., and K. Ramamritham, editors. 1988. *Hard Real-Time Systems.* IEEE Computer Society Press.

Stankovic, J. A., and K. Ramamritham, editors. 1993. *Advances in Real-Time Systems.* IEEE Computer Society Press.

Stankovic, J. A., and K. Ramamritham. 1989. The Spring Kernel: A new paradigm for real-time operating systems. *Special Issue on Real-Time Operating Systems, ACM Operating System Review*, 23(3):54–71.

Strosnider, J. K., and D. Katcher. 1991. Bridging the gap between scheduling theory and reality. *Office of Naval Research, Fourth Annual Review and Workshop, Foundations of Real-Time Computing.*

Strosnider, J. K., and T. E. Marchok. 1989. Responsive, deterministic IEEE 802.5 token ring scheduling. *Journal of Real-Time Systems*, 1(2):133–158.

Strosnider, J. K., T. Marchok, and J. Lehoczky. 1988. Advanced real-time scheduling using the IEEE 802.5 token ring. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 42–52.

Strosnider, J. K. 1988. *Highly Responsive Real-Time Token Rings.* PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University.

Tindell, K. 1993. Holistic schedulability analysis for distributed hard real-time systems. Technical Report YCS 197, Department of Computer Science, University of York.

Tanenbaum, A. S. 1988. *Computer Networks*. Prentice-Hall, Inc., 2nd edition.

Teener, M., and R. Gvozdanovic. 1989. FDDI-II operation and architecture. In *Proceedings of the IEEE Conference on Local Computer Networks*, pages 49–61.

Tobagi, F. A., F. Borgonovo, and L. Fratta. 1983. ExpressNet: A high performance integrated-services local area network. *IEEE Transactions on Selected Areas in Communications*, SAC-1(5):898–913.

Tokuda, H., C. W. Mercer, Y. Ishikawa, and T. E. Marchok. 1989. Priority inversions in real-time communication. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 348–359.

Trajković, L., and S. J. Golestani. 1992. Congestion control for multimedia services. *IEEE Network*, 6(5):20–26.

Ulm, J. N. 1982. A timed token ring local area network and its performance characteristics. In *Proceedings of the IEEE Conference on Local Computer Networks*, pages 50–56.

U.S. Department of Defense. 1994. *Survivable Adaptable Fiber Optic Embedded Network (SAFENET)*. MIL-STD-2204A.

van Tilborg, A. M., and G. M. Koob. 1991a. *Foundations of Real-Time Computing: Formal Specifications and Methods*. Kluwer Academic Publishers.

van Tilborg, A. M., and G. M. Koob. 1991b. *Foundations of Real-Time Computing: Scheduling and Resource Management*. Kluwer Academic Publishers.

Vetterli, M., and M. W. Garrett. 1990. Joint source channel coding for real time packet services. In *Proceedings of the Australian Video Communications Workshop*, pages 136–145.

Volz, R. A., L. Sha, and D. Wilcox. 1991. Maintaining global time in Futurebus+. *The Journal of Real-Time Systems*, 3(1):5–17.

Welch, J. L., and N. Lynch. 1988. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36.

Wilkov, R. S. 1972. Analysis and Design of Reliable Computer Networks. *IEEE Transactions on Communications*, COM-20(6):660–678.

Yamamoto, Y., and K. Takahashi. 1989. ATM network performance design covering video communications. In *Proceedings of the 4th ATERB Fast Packet Switching Workshop*.

Yates, D., J. Kurose, D. Towsley, and M. G. Hluchyj. 1993. On per-session delay distributions and the call admission problem for real-time applications with QOS requirements. In *Proceedings of ACM SIGCOMM'93*, pages 2–12.

Yao, L.-J. 1994. *Real-time communications in token ring networks*. PhD thesis, Department of Computer Science, University of Adelaide.

Yao, L.-J., and W. Zhao. 1991. Performance of an extended IEEE 802.5 protocol in hard real-time systems. In *Proceedings of IEEE Infocom'91*, pages 469–478.

Yau, S, editor. 1989. Special issue on distributed computer systems. *IEEE Transactions on Computers*, 38(8).

Yemini, Y. 1983. A bang-bang principle for real-time transport protocols. In *Proceedings of ACM SIGCOMM'83*, pages 262–268.

Zafirovic-Vukotic, M., I. G. Niemegeers, and D. S. Valk. 1988. Performance analysis of slotted ring protocols in HSLAN'S. *IEEE Transactions on Selected Areas in Communications*, 6(6):1011–1024.

Zhang, H., and D. Ferrari. 1993. Rate-controlled static priority queueing. In *Proceedings of IEEE Infocom'93*, pages 227–236.

Zhang, H., and S. Keshav. 1991. Comparison of rate-based service disciplines. In *Proceedings of ACM SIGCOMM'91*, pages 113–121.

Zhang, L. 1990. Virtual clock: A new traffic control algorithm for packet switching networks. In *Proceedings of ACM SIGCOMM'90*, pages 19–29.

Zhao, W, editor. 1989. Special issue on real-time operating systems. *ACM Operating Systems Review*, 23(3).

Zhao, W., C. Barter, and N. Malcolm. 1989. Virtual time CSMA protocols with two version message model for real-time communications. In *Proceedings of the 1989 IEEE International Conference on Networks*, pages 295–300.

Zhao, W., and K. Ramamritham. 1986. A virtual time CSMA protocol for hard real-time communications. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 120–127.

Zhao, W., and K. Ramamritham. 1987. Virtual time CSMA protocols for hard real-time communications. *IEEE Transactions on Software Engineering*, SE-13(8):938–952.

Zhao, W., and J. A. Stankovic. 1989. Performance analysis of FCFS and improved FCFS scheduling algorithms for dynamic real-time computer systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 156–165.

Zhao, W., J. A. Stankovic, and K. Ramamritham. 1988. A multi-access window protocol for time constrained communications. In *Proceedings of the 8th IEEE International Conference on Distributed Computing Systems*, pages 384–392.

Zhao, W., J. A. Stankovic, and K. Ramamritham. 1990. A window protocol for transmission of time constrained messages. *IEEE Transactions on Computers*, 39(9):1186–1203.

Zheng, Q., and K. G. Shin. 1992. Fault-tolerant real-time communication in distributed computing systems. In *Digest of Papers, FTCS-22*, pages 86–93.

Zheng, Q., and K. G. Shin. On the ability of establishing real-time channels in point-to-point packet-switched networks. To appear in IEEE Transactions on Communications.

Zheng, Q., and K. G. Shin. 1993. Synchronous bandwidth allocation in FDDI networks. In *Proceedings of ACM Multimedia'93*, pages 31–38.

Ziegler, J. 1989. MIL-STD-1553 takes over serial data bus designs. *Computer Design*, 28(17):99–102.

Zimmermann, H. 1981. OSI Reference Model — the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communication*, COM-28(4):425–432.

Znati, T. 1991. Deadline Driven Window Protocol for Transmission of Real-Time Traffic. In *Proceedings of the 10th IEEE International Conference on Computers and Communications*, pages 667–673.