Real-time Communication in Multi-hop Networks

Dilip D. Kandlur, Kang G. Shin

Real-Time Computing Laboratory
Department of Elec. Engr. and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109–2122.
email: kgshin@alps.eecs.umich.edu

Domenico Ferrari

Computer Science Division
EE & CS Department
University of California
Berkeley, CA 94720.

ABSTRACT

Communication in real-time systems has to be predictable, because unpredictable delays in the delivery of messages can adversely affect the execution of tasks dependent on these messages. In this paper, we develop a scheme for providing predictable inter-process communication in real-time systems with (partially connected) point–to–point interconnection networks, which provides guarantees on the maximum delivery time for messages. This scheme is based on the concept of a real-time channel, a unidirectional connection between source and destination. A real-time channel has parameters which describe the performance requirements of the source–destination communication, e.g., from a sensor station to a control site. Once such a channel is established, the communications subsystem guarantees that these performance requirements will be met. In this paper, we concentrate on methods to compute guarantees for the delivery time of messages belonging to real-time channels. We also address problems associated with allocating buffers for these messages and develop a scheme which preserves delivery time guarantees.

Index Terms — Real-time systems, communication, scheduling, guaranteed delay, point–to–point networks.

The work reported here is supported in part by an IBM Graduate Fellowship and by the Office of Naval Research under Contract N00014–85–K–0122. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

1 Introduction

Due to their potential for high perfomance and high reliability, multicomputer systems with point–to–point interconnection networks are natural candidates for time- and mission-critical applications. The presence of multiple disjoint paths between nodes in these networks makes them robust to link and node failures. Also, since it is possible to support simultaneous transmissions in different links, the total throughput achievable is higher. Examples of this type of network include hypercubes and meshes. The problem of providing predictable inter-process communication is of great significance in real-time systems because unpredictable delays in the delivery of messages can affect the completion time of tasks participating in the message communication. If these tasks have deadlines associated with them, communication delays could cause the tasks to miss their deadlines. In this paper, we will address the issue of guaranteeing the delivery of messages with time constraints in multicomputers with point–to–point interconnection networks.

The problem of time-constrained communication has been studied by several researchers, since it plays an important role in video– and voice– data transmission over a data network. Recently, it has also been studied in the context of communication in embedded or real-time systems. These efforts have been directed mainly towards designing medium access protocols for multiple-access networks which consider time constraints on messages. The medium access protocols try to implement a distributed scheduling discipline, and the focus has been on developing algorithms which try to minimize message loss. For example, in the case of CSMA/CD networks, the proposed algorithms include the virtual-time based methods [1] and the window based access methods [2, 3]. The survey paper by Kurose *et al.* [4] discusses many of the other proposed techniques. Most of these schemes can be classified as *best-effort* schemes, where the system tries to ensure that most messages meet their deadlines, but it cannot give any guarantees about the delivery times. The performance of these algorithms is demonstrated mainly by simulation using certain stochastic arrival and deadline patterns. On the other hand, when the system has some information about the arrival pattern of messages, it can try to give guarantees about their delivery times. For example, Strosnider and Marchok [5] use a variation of the rate-monotonic scheduling algorithm to control access to a token-ring network. They assign priorities at design time to message sources based on the periodicity of message generation and they can check for the possibility of deadline overrun at that time. In their work, the deadline of each periodic message is implicitly assumed to be the beginning of the next period.

The network under consideration here is, however, not a multiple-access network, but has a point–to–point interconnection structure which, as mentioned earlier, has potential for higher performance and reliability than bus/ring structures. In this case, the problem is more complicated than multiple–access networks, because we have to consider delivery time constraints across multiple stages in the network. In this type of network, there is only one source node for any network link, so the issue to be addressed is not one of access to the medium but that of message scheduling in the network nodes. Although it is possible that messages which have to traverse multiple links to reach the destination may suffer from the problem of higher latency, the latency can be made more *predictable* using message scheduling and network flow control. The higher latency need not be a problem for periodic messages, where it is generally assumed that the deadline for a periodic message is the beginning of the next period, which is typically large. There has not been much work reported on the problem of providing time-constrained communication in a local-area point–to–point interconnection network. However, the work by Ferrari, Anderson and others in the DASH and Tenet projects [6, 7, 8, 9] dealing with continuous–media communication in wide-area networks is closely related to the work reported here. (See Section 6 for the differences between this work and ours.)

In this paper, we will be concentrating on issues related to the design of time-constrained communication in which the system provides *a priori* guarantees for message delivery. We will also briefly discuss how best-effort delivery can easily be accommodated into the design. Section 2 describes the computing environment and defines the problems to be addressed. In Section 3 we present a solution scheme which is general and works for messages of arbitrary length. Section 4 describes a buffer management scheme which is consistent with the message handling scheme descibed in Section 3. Section 5 presents extensions to this scheme to help it handle long messages more effectively. We discuss other related work in Section 6 and draw our conclusions in Section 7.

2 Background and Problem Statement

The intended application for this work is in embedded real-time systems, and thus, we make assumptions appropriate for that environment. The system consists of processor nodes connected by point–to–point dual simplex links. A node in the system can have several incoming and outgoing links connected to it, and these links can operate in parallel (as in [10]). The nodes run a common distributed real-time operating system which is responsible for control of the network. The operating system also synchronizes clocks and maintains a global time-base for the system, where the maximum skew between clocks on different nodes is very small compared to the end–to–end message delivery delay. A scheme for achieving this synchronization is presented in [11].

Communication between user-level entities in this system can either be connection-oriented or connectionless. In the connection-oriented case, it can be either message stream or byte stream oriented. In the context of time-constrained communication, the communication model has to preserve message boundaries, and so a byte stream model is not suitable. Connection-oriented service is considered more suitable for applications which require guaranteed delivery time for communication [7]. In order to make a guarantee about delivery time for a message, the system requires information about other message sources that can contend for resources with this message. It is therefore necessary to provide a mechanism for describing the characteristics of communication, so that resources can be reserved for real-time connections. The connection establishment procedure gives the service provider, in this case the distributed operating system, the opportunity to reserve resources for the connection, and for the user to specify its requirements. Therefore, the abstraction that we use for guaranteed time-constrained communication is one of connection-oriented sequenced messages, which we call a *real-time channel* or simply *channel.* In a bidirectional connection between a pair of user entities, the message generation characteristics may differ substantially for the two directions. Hence, it is preferable to restrict the real-time channel to unidirectional communication, and a bidirectional connection can be composed from a pair of channels.

The resources required by a channel include network bandwidth, buffer space, and message processing bandwidth. The operating system can make resource reservations based on the communication requirements of the user, as specified in the request for the connection. These requirements consist of the source and destination end-points, a description of the message generation process, and the desired end–to–end guarantee for the delivery time. Without a description of the message generation process, the service provider cannot compute the resource requirements and hence it cannot provide a guarantee.

It is perceived that a significant portion of the traffic which requires guarantees will be periodic communication, so the model for message generation is based on this perception. The message generation process is specified in terms of a linear bounded arrival process, a model which was first proposed by Cruz [12]. This model has also been adopted by Anderson and others [13, 14] for continuous media applications, and some of the terminology given below is from [13]. The arrival process has the following parameters:

maximum message size	S_{max}	(bytes)
maximum message rate	R_{max}	(messages/second)
maximum burst size	B_{max}	(messages)

The model includes the restriction that, in any time interval of length t, the number of messages generated may not exceed $B_{max} + t \cdot R_{max}$, and that the length of each message may not exceed S_{max}. R_{max} is a bound on the message generation rate and its reciprocal, I_{min}, is the minimum

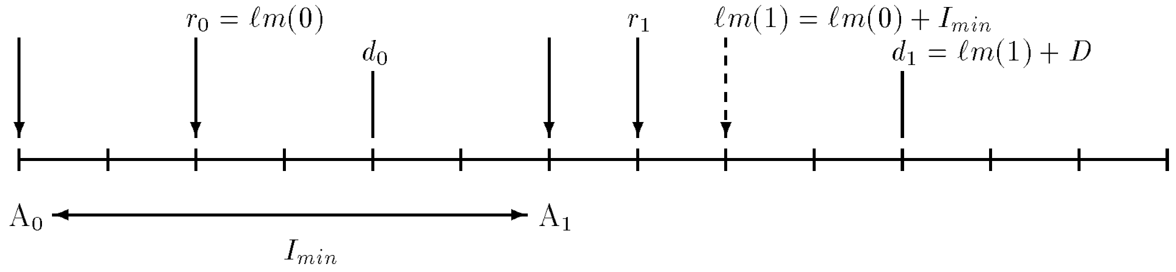

Figure 1: Delivery time guarantee.

(logical) inter-arrival time between messages. The burst parameter B_{max} puts a bound on the allowed short-term variation in the message generation rate, and partially determines the buffer space requirement of the channel. Message generation which is not periodic can be represented in this model using an estimate of the worst-case inter-arrival time and the average rate of generation.

The semantics of the end–to–end guarantee on delivery time are also based on this model. The *logical generation time*, $\ell(m)$, for a message m can be defined as

$$\begin{aligned} \ell(m_0) &= t_0 \\ \ell(m_i) &= max\{\ell(m_{i-1}) + I_{min},\ t_i\} \end{aligned}$$

where t_i denotes the actual generation time of m_i. If d is the end–to–end delay for the channel, the system *guarantees* that any message m_i will be delivered to the destination node by time $\ell(m_i) + d$. In other words, when the inter-arrival time between messages is at least I_{min}, the system guarantees that each message in the channel incurs a delay of at most d seconds since its logical arrival time. However, messages which arrive in a burst, where the inter-arrival time is less than I_{min}, may suffer a larger delay since the guarantee is given with respect to the logical arrival time. This possible increase in delay is a consequence of regulation: arrivals at each node have to be regulated in order to prevent burst arrivals on one channel from affecting the guaranteed delay of messages belonging to other channels.

This definition of the message delivery guarantee is well suited for periodic message generation. For example, consider a periodic task which generates a message at the end of its execution. The generation time of the message depends upon the *response time* seen by the task, and this can vary from one invocation to another. If these messages are transmitted on a real-time channel, the worst-case delivery time of a message can be linked to the *arrival time* of the periodic task. Figure 1 shows two instances of a periodic task which occur at time A_0 and A_1 respectively. The messages form a part of a real-time channel with inter-arrival time

I_{min} and delay D. The first instance of the task experiences its worst-case response time R, so the message is released at time r_0. The second instance of the task experiences a response time $r_1 - A_1 < r_0 - A_0$. Hence, the interval between the generation time of the first and second messages is less than I_{min}, i.e., the second message arrived early. The logical generation time $\ell m(1)$ assigned to the second message is therefore greater than its actual arrival time. However, it can be seen that the generated messages will always be delivered by time $R + D$ from the beginning of the task period where R is the worst-case execution time of the periodic task. This property can be used in any off-line schedulability analysis for the system.

This paper deals with problems connected with computing a guaranteed end–to–end delay for messages belonging to a channel, and the scheduling of messages to achieve this goal. These two problems are related because it is necessary to understand the scheduling environment in order to compute a guarantee. In the next section, we will first present a simple channel establishment procedure in which the problem of computing an end–to–end guaranteed delay is reduced to the problem of guaranteeing the worst-case delay for a single station/node.

3 Basic Solution Approach

There are two distinct phases in handling real-time channels: channel establishment and run-time message scheduling. The channel establishment phase is outlined in Figure 2, and begins with the selection of a route for the channel. There are two alternatives for routing packets in the network: dynamic routing and static routing. Dynamic routing offers the advantage that it can adapt to the network load and can reduce the average delivery delay for messages. However, it is very difficult to make any guarantees on message delivery for a channel based on dynamic routing, in which a message can use one of several alternate routes through the network. Therefore, we use static routing for messages belonging to real-time channels. We will assume that a route has been selected for the channel, possibly using a scheme like the one presented in [15], and concentrate on the computation of guarantees and on buffer management.

The worst-case delay for a message at a link depends upon the scheduling algorithm used and the other channels which use the link. The link may also be used by messages without time constraints, but the effect of these messages on the worst-case delay is limited (i.e., restricted to a single packet delay if packet transmission cannot be aborted while in progress) because they would belong to a lower priority class. Hence, we can restrict our attention to messages belonging to real-time channels while calculating the worst-case delay. The scheduling environment that we encounter in message scheduling is one of independent, possibly periodic, message arrivals which have deadlines associated with them. The message deadline may be related to its

1. Select a source–destination route for the channel.
2. Compute the worst-case delay for a message on each link on the route. In this computation, it is necessary to ensure that the new channel does not affect the guaranteed delivery times of existing channels. Also compute the buffer requirement for this channel.
3. Compute the sum of the worst-case delays which were determined in Step 2, and check whether it is less than the user-specified delay. This is the channel establishment test.
4. If the channel can be established, divide the user-specified delay among the links on the route based on their worst-case delay for the message. Adjust the buffer requirements based on these allocated delays.

Figure 2: **Channel Establishment Procedure**

period, but it is not necessarily the beginning of the next period. There are several approaches to scheduling these messages, which can be categorized as fixed priority or dynamic priority algorithms. For example, Earliest Due Date [16] is a dynamic priority algorithm, whereas rate monotonic scheduling [17] is a fixed priority algorithm. We will now discuss the problems associated with the use of these algorithms for message scheduling and channel establishment.

Deadline Scheduling: The Earliest Due Date (EDD) algorithm schedules messages in the order of their deadlines, with higher priority given to messages which have closer deadlines. Liu and Layland [17] have shown that EDD is optimal for preemptive scheduling of periodic tasks when the task deadline is equal to the beginning of its next period, and that a feasible schedule exists whenever the total utilization is less than one. However, there is no similar result (based on utilization alone) available when the task deadlines are not related to their periods. The main drawback of EDD scheduling is that computation of guarantees is difficult, since the priority of a task depends upon the relative order of arrival of the tasks. A multi-class version of the EDD algorithm has been used for scheduling real-time messages by Ferrari and Verma [7, 6], who also present sufficient conditions for the existence of feasible schedules. Their approach will be discussed in greater detail in Section 6.

Fixed Priority Scheduling: Scheduling decisions can be based on a fixed (static) priority scheduling algorithm where messages are processed and transmitted in the order of priority. For example, in rate-monotonic scheduling the priority assigned to a channel is related to the frequency of occurrence of messages on that channel. For any priority assignment scheme, if message arrivals on all the channels are assumed to be strictly periodic, we can determine whether the set of channels is schedulable. This is done by computing the worst-case response

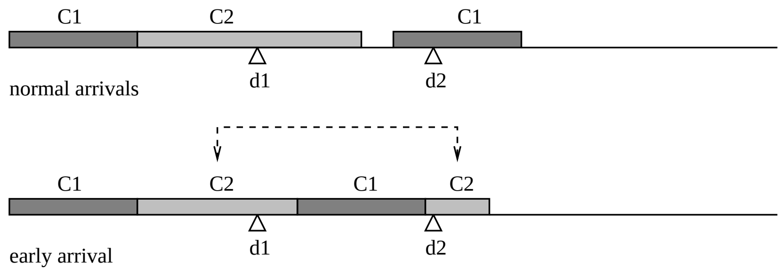

Figure 3: Effect of early arrivals.

time for each message using a *critical time zone* analysis similar to the one used by Liu and Layland [17], and verifying that the worst-case response time is less than the delay assigned to that channel. The details of this scheme for analysis can be found in Appendix A.

There are some problems with priority based scheduling and the computation of guarantees when we consider multiple stage service, like service for a message which has to traverse multiple links. The response time for the message varies depending upon the arrival time of other messages at a node. Therefore, even if messages are generated with a fixed inter-arrival time at the source, the inter-arrival time for the message at the next service point is not constant. Early arrivals are also possible due to burstiness in the message generation at the source. A message which arrives early at a node can cause a lower priority message to miss its deadline. Figure 3 shows the results of preemptive priority scheduling for two message streams, M_1 and M_2, with service requirements C_1 and C_2, where M_1 has higher priority than M_2. The figure shows how the early arrival of a message on M_1 can cause a message on M_2 to miss its deadline d_2. Note that a similar effect can be observed even when a non-preemptive discipline is used. One solution for this problem is to "hold back" the early arrival and not consider it for transmission until its scheduled arrival time. However, this scheme involves the setting and resetting of timers and is expensive to implement. Also, it implies that the message cannot make the best progress possible on lightly loaded links.

3.1 Proposed Algorithm

The discussion on the problems of deadline and fixed priority scheduling in a multi-hop communication system suggests the use of a combination of deadline and fixed priority scheduling. Our channel establishment scheme uses fixed priority scheduling for computing the delay, but we use a form of EDD as the run-time message scheduling algorithm. When a channel is to be established, for each link on its route, we estimate the worst-case response time for a message based on fixed-priority scheduling using the method outlined in Appendix A. The response time analysis is based on an approximation of preemptive scheduling. Pure preemptive scheduling cannot be used in the context of message scheduling, because, if the transmission of a message is interrupted, the message is lost and has to be retransmitted. To achieve the benefits of preemptive scheduling, the message has to be split into packets so that message transmission can be interrupted at the end of a packet transmission, without loss. (This is analogous to allowing an interrupt at the end of an instruction execution.) Therefore we consider a message to be a set of one or more packets, where the packet size is bounded. Packet transmission is non-preemptive, but *message* transmission can be considered to be preemptive.

The priority assigned to the new channel at a link depends upon the characteristics of the other channels going through the link. The total response time, which is the sum of the response times over all the links on the route of the channel, is checked against the maximum permissible message delay and the channel can be established only if the latter is greater. In this case, the permissible message delay is split proportionally among the different links. While using this procedure, it is necessary to ensure that the new channel does not affect the guaranteed delivery times of existing channels. This is taken care of in the priority assignment algorithm.

3.2 Priority Assignment

The worst-case response time computation for a new channel at a link on the route requires an assignment of priorities for existing channels that use this link. This priority assignment problem can be defined formally as follows. Let $\{M_i = (C_i, p_i, d_i), i = 1, \ldots, k\}$ be the set of k existing channels through a link ℓ, where C_i is the maximum service time required for messages of channel M_i on this link, $p_i = I^i_{min}$ is the message inter-arrival time, and d_i is the permissible delay which has been assigned to the channel for this link. The service time requirement C_i is proportional to the product, $S^i_{max}.R^i_{max}$, of the maximum message size and the maximum message rate for the channel M_i. Given a new channel $M_{k+1} = (C_{k+1}, p_{k+1})$ to be established (the delay bound need not be specified on a per-link basis), we have to find a priority assignment for this augmented set of channels such that the response time r'_i computed for a channel under this priority assignment satisfies the constraint $r'_i \leq d_i, i = 1, \ldots, k$. An assignment which satisfies this constraint is called a *feasible priority assignment*. Moreover, we would like to find a feasible priority assignment which will minimize the response time r'_{k+1} for the new channel M_{k+1}. By finding the minimum feasible response time on each link, we can improve the chances of satisfying the channel establishment test.

Consider the procedure D_Order, shown in Figure 4, which assigns priorities and computes the response time for all channels, including the one to be established. This procedure works

1. Arrange the channels in ascending order of their associated delay d_i.
2. Assign the highest priority to the new channel M_{k+1}. Assign priorities to the other channels based on this order, with high priority assigned to channels with small delays.
3. Compute the new (worst-case) response times r'_i for the existing channels based on this priority assignment.
4. In the priority order, find the smallest position q such that $r'_i \leq d_i$ for all channels with position greater than q.
5. Assign priority q to the new channel and compute the response time r'_{k+1}.

Figure 4: **Assignment Procedure D_Order**

under the condition that, for all channels, $d_i \leq p_i$. That is, the worst-case delay at each link for any channel does not exceed its inter-arrival time. Note that the total end–to–end delay can exceed the inter-arrival time of the channel.

We can prove that, when $d_i \leq p_i, \forall i$, assignment procedure D_Order is optimal in the sense that the computed response time r'_{k+1} is the minimum possible for any feasible priority assignment. Lemma 1 proves that the priority assignment based on delays is optimal in the sense that if there is any feasible priority assignment for the channel set, there is a feasible priority assignment based on delays. This is in fact a generalization of the optimality result for rate-monotonic scheduling [17]. In their model, Liu and Layland assumed that the deadline for an instance of a periodic task is the release time of the next instance, that is, $d_i = p_i$. The rate-monotonic algorithm assigns priorities to periodic tasks based on their periodicity, with high priority assigned to tasks with high frequency. The optimality result for this scheduling algorithm can be derived from Lemma 1 by substituting $d_i = p_i, \forall i$. It has been brought to our attention that a result similar to Lemma 1 has also been proved by Leung and Whitehead [18].

Lemma 1: Consider the set of channels $\{M_i = (C_i, p_i, d_i), i = 1, \ldots, k\}$ through a link. Assume that there exists a feasible fixed priority assignment for the channels such that the computed response time for each channel satisfies the constraint $r_i \leq d_i \leq p_i, \forall i$. Then, the priority assignment P^D, based on an increasing order of delays with high priority assigned to channels with small delays, is also a feasible priority assignment.

Proof: We can assume, without loss of generality, that the set of channels is ordered by priority in accordance with a feasible priority assignment P. That is, $i < j$ implies that M_i has higher priority than M_j under P. Now, if $d_i \leq d_j, \forall i < j$, our proof is complete. Otherwise, there exists at least one pair of adjacent channels, M_i and M_{i+1} such that $d_i > d_{i+1}$. In this case, we want to prove that priorities of these two channels can be swapped to yield another feasible assignment, P'.

Let r_i and r_{i+1} be the worst-case response times of the two channels M_i and M_{i+1}, computed under the priority assignment P using the scheme shown in Appendix A. Then, since $d_{i+1} < d_i \leq p_i$ and $r_{i+1} \leq d_{i+1}$, $r_{i+1} < p_i$, and so r_{i+1} includes the service time for exactly one instance of M_i. Under P', the response time for M_{i+1} is reduced, so the constraint $r'_{i+1} \leq d_{i+1}$ is trivially satisfied. Also, by the definition of the worst-case response time (see Eq. (A.2) in Appendix A), r_{i+1} contains exactly one instance of M_{i+1}. The response time r'_i for M_i under P' also has to contain one instance of M_i and M_{i+1}, and it can be seen that $r'_i = r_{i+1} \leq d_i$.[1] For all other channels we have $r'_j = r_j \leq d_j$, and hence P' is a feasible priority assignment.

It is evident that by a sequence of pairwise exchanges, we can reach P^D and show that it is a feasible assignment. ■

Theorem 1: Assignment procedure D_Order yields the smallest response time for the new channel, M_{k+1}.

Proof: Suppose there is some other feasible priority assignment P which yields a smaller response time. If r_{k+1} is the response time for M_{k+1} under P, we can set $d_{k+1} = r_{k+1}$. Consider any channel M_j which has lower priority than M_{k+1}. Then, $r_{k+1} < r_j \leq d_j$ and hence $d_{k+1} < d_j$. Now, by Lemma 1, the priority assignment P^D based on the channel delays is also a feasible assignment. Under this assignment, $d_{k+1} < d_\ell$ implies that M_{k+1} has priority over M_ℓ. Therefore, any channel which has priority lower than M_{k+1} under P also has priority lower than M_{k+1} under P^D. Also, a channel which has priority higher than M_{k+1} under P could have priority lower than M_{k+1} under P^D. This means that the response time for M_{k+1} under P^D will be no greater than the response time under P. Contradiction. ■

D_Order can be implemented efficiently by storing some additional information about the existing channels. We can maintain the channels in ascending order of permissible delays, so the cost of sorting can be ameliorated. The computation of the response time for all channels is then the most important operation. We can eliminate duplicate computations by computing the response times in ascending order. For example, $r_{i+1} \geq r_i + C_i$, and we can use this as the starting point in the scan. Also, in the computation, if $r'_i > d_i$ then the computation for r'_i can be aborted.

[1] Note that this result holds only because $r_{i+1} \leq p_{i+1}$, so that when we switch priorities only one instance of M_{i+1} is included in the response time for M_i.

In the last stage of the channel establishment procedure, delays will be assigned to the new channel, for each link on the source–destination route. For any link, the assigned delay for that link, d_{as}, has to be such that $d_{as} \geq r'_{k+1}$. It can be shown that if channels are allotted priorities based on this assigned delay, the resulting priority order is still feasible. The new priority order for the channels is not necessarily the one used to compute the response time r'_{k+1} because there may exist a channel with $r'_{k+1} \leq d_i < d_{as}$.

3.3 Run-time Scheduling

A node in the system can have several incoming and outgoing links connected to it. These links can operate in parallel, so each outgoing link is considered as a separate entity for scheduling. Messages are composed of packets, and packets carry information about the message, and the channel, to which they belong. When a packet arrives at a node, or is generated in the node, it is dispatched to the appropriate outgoing link. Suppose the arriving packet belongs to the ith message on the channel M_c. All packets belonging to the ith message would be assigned the same *logical arrival time*, $\ell_c(m_i)$, which is the logical arrival time for the message. For the source node s of channel M_c, $\ell_{c,s}(m_i)$ is defined exactly in the same way as the logical generation time in Section 2, where t_i denotes the generation time of message m_i.

$$\begin{aligned} \ell_{c,s}(m_0) &= t_0 \\ \ell_{c,s}(m_i) &= max\{(\ell_{c,s}(m_{i-1}) + I^c_{min}),\ t_i\}. \end{aligned} \tag{3.1}$$

For nodes other than the source node, the logical arrival time at the node is based on the logical arrival time of the message at the upstream node. The information about the logical arrival time is carried in the packet, as a part of the packet header. Consider two adjacent nodes a and b, sharing a link (a,b) which is a part of the route for channel M_c. The logical arrival time for m_i at node b, $\ell_{c,b}(m_i)$, is defined as

$$\ell_{c,b}(m_i) = \ell_{c,a}(m_i) + d_{c,a} \tag{3.2}$$

where $d_{c,a}$ is the worst-case delay for messages on channel M_c at node a. It can be seen that the logical arrival time of a message at any node is ultimately based on its logical arrival time at the source node. This definition of the logical arrival time is feasible because the nodes in the system have synchronized clocks and the maximum skew between clocks on different nodes is small compared to the message delivery delays [11].

Alternatively, it is conceivable to define $\ell_{c,b}(m_i)$ based only on the *observed* arrival time of messages at node b, without carrying any timestamps in the packet headers. However, this type of scheme has problems when we consider multiple-packet messages. For example, $\ell_{c,b}(m_i)$ can be defined based on the arrival time of the *first* packet of m_i. But then, it is possible that the last packet of m_i may arrive up to (approximately) $d_{c,a}$ time units later than the first packet, if we consider the worst-case delay for m_i over just one link (a, b). When we consider delays for m_i over multiple links, the separation between the first and last packets can increase further. In this case, the last packet can be assigned a logical arrival time which is *beyond* its local deadline. This can result in some problems for the deadline scheduler. On the other hand, if $\ell_{c,b}(m_i)$ is defined based on the arrival time of the *last* packet of m_i, the message cannot be processed until the last packet has arrived. Moreover, it would be necessary to guard against loss of the last packet of a message, possibly using message timers.

The logical arrival time that is assigned to messages is used by the message scheduler, which uses a variation of the multi-class EDD algorithm. In the following description, the second subscript has been dropped from the notation for the logical arrival time, since we are dealing with a single node. The scheduler maintains three queues for each outgoing link, corresponding to three service classes.

Queue 1 Packets belonging to real-time channels with $\ell_c(m_i) \leq$ current_time, arranged in the order of increasing deadlines.

Queue 2 Other packets arranged in the order of increasing deadlines.

Queue 3 Packets belonging to real-time channels with $\ell_c(m_i) >$ current_time, arranged in the order of increasing logical arrival time.

Queue 1 and Queue 3 contain packets which belong to real-time channels, while Queue 2 contains all other types of packets. Queue 1 contains *current* real-time packets, which have to be scheduled in the order of their deadline, hence it is organized by increasing packet deadlines. Current real-time packets are those packets whose logical arrival time is less than the current clock time at the node. Queue 2 contains packets for which no guarantees are given, but which can have deadlines associated with them. For example, packets belonging to messages which request "best-effort" type service would fall in this category. The service provided for this class of packets is improved by giving them priority over real-time packets which are not *current*. Since they are not pertinent to the current discussion, we will not elaborate on the treatment of this class any further. Packets in Queue 3 are those which have arrived early, either because of burstiness in the message generation or because they encountered delays which were smaller than the budgeted worst-case delays at some upstream nodes. These packets are stored in the order of their logical arrival time because they have to be transferred to Queue 1 as they become current.

1. find $\ell_c(m_i)$, the logical arrival time of the message to which this packet belongs.
2. set $\ell t(\text{packet}) = \ell_c(m_i)$.
3. set the deadline for this packet to $\ell_c(m_i) + d_c$.
4. **if** ($\ell t(\text{packet}) \leq$ current_time)
   insert packet into Queue 1
   else
   insert packet into Queue 3
5. invoke the dispatcher.

Figure 5: **Processing on Packet Arrival**

The actions taken when a real-time packet arrives are shown in Figure 5, whereas non real-time packets are simply inserted into Queue 2. The logical arrival time is determined based on the channel to which the packet belongs and the sequence number of the message of which it is a part. This is the logical time, $\ell t()$, of a packet. The deadline for the packet is set to $\ell_c(m_i) + d_c$, where d_c is the worst-case delay guaranteed for channel M_c at this node. Since all packets in a message have the same $\ell_c(m_i)$ and the same d_c, they will all have the same deadline. The packet is then inserted into Queue 1 or Queue 3, depending upon whether it is current or early. Lastly, the dispatcher is invoked.

The dispatcher, which is shown in Figure 6, first checks whether any real-time packets in Queue 3 have become current and transfers such packets to Queue 1. If the link is idle, it examines the queues in the order of priority looking for a packet to transmit. Packets in Queue 3 are considered for transmission only if their logical time is within the *horizon*, H_k, for the link. The horizon is link-dependent and is used for flow control, as explained in Section 4. These packets are scheduled in the order of logical time, primarily because they are queued in that order. Also, this ordering makes it easy to identify packets which are within the horizon (and can be considered for transmission). The dispatcher is also invoked upon completion of transmission of a packet on the link. In some situations, it is possible that Queues 1 and 2 are empty and Queue 3 only contains packets which are ineligible for transmission. In such cases, a timer can be used to trigger the dispatcher at the appropriate future instant.

Proof of Correctness

We now have to establish that this run-time scheduling scheme will conform with the guarantees computed using fixed message priorities. This is stated formally in Theorem 2 below.

```
Examine Queue 3. Transfer those packets which have ℓt(packet) < current_time, to Queue 1.
if (link idle)
    if (Queue 1 nonempty)
        start transmission of head(Queue 1)
    else if (Queue 2 nonempty)
        start transmission of head(Queue 2)
    else if (Queue 3 nonempty)
        if (ℓt(head(Queue 3)) < current_time + H_k)
            start transmission of head(Queue 3)
        end
    end
end
```

Figure 6: **Dispatcher**

Theorem 2: Consider a link containing a set of real-time channels which satisfy the following properties.

1. The channels are assigned delays for this link using the procedures in Section3.1 (Figures 2 and 4.
2. Arriving messages are assigned logical arrival times using Eqs. (3.1) and (3.2).
3. Messages are scheduled using the procedures in Figures 5 and 6.

Then, all messages arriving on real-time channels meet their delay requirements on the link.

Proof: We approach this in two stages. In the first stage, we show that if we use this scheduling scheme on a channel set (which is schedulable) that has strict periodic arrivals satisfying the inter-arrival time constraints, then no message will miss its deadline. In the second stage we consider the effects of deviations from the periodic behavior and show how our scheme can accommodate them.

Stage 1: The system times for channels have been computed based on priority scheduling, taking into account the maximum rate of arrival of messages; and these times satisfy the local deadlines for each channel. Hence, when arrivals are all periodic and conform to the inter-arrival time constraints, a feasible schedule based on priorities exists. Dertouzos [16] has shown that this implies the existence of a feasible EDD schedule for this message set, which is based on the deadlines assigned to the channels. In this case, the logical arrival time of each message is the same as its arrival time and all arrivals go into Queue 1. Since this queue is scheduled using the EDD algorithm, and a feasible EDD schedule is possible, the delay requirements of all messages will be satisfied.

Stage 2: Consider a message m_j belonging to channel M_c, which arrives at the source node before its logical arrival time, that is, $t_{arr,m_j} < \ell_c(m_j) = \ell_c(m_{j-1}) + I^c_{min}$. This message would be assigned a deadline, $D_j = \ell_c(m_j) + d_c$, corresponding to its logical arrival time and would be inserted into Queue 3. There are two possibilities to consider: 1) m_j is transmitted directly out of Queue 3, and 2) m_j is subsequently transferred to Queue 1. In the first instance, we can conclude that m_j was transmitted before time $\ell_c(m_j)$ and hence it was transmitted before its deadline. In the second case, some packets of m_j were transferred to Queue 1. Since Queue 3 is scanned at the end of every packet transmission, this transfer will occur by time $\ell_c(m_j) + C_{pkt}$, where C_{pkt} is the transmission time for a single maximum-length packet. By the argument given in Stage 1, we can assert that, if the message m_j arrived into the system at its logical arrival time, a feasible EDD schedule exists by which all messages will meet their deadline. Note that the computed response time for the channels includes C_{pkt}. Hence, the packets of m_j which were transferred to Queue 1 will be transmitted before their deadline. The same argument can be applied even when multiple messages arrive prior to their logical arrival time.

If a message m_i arrives late at the source node, that is, $t_{arr,m_i} > \ell_c(m_{i-1}) + I^c_{min}$, it will be assigned a logical arrival time $\ell_c(m_i) = t_{arr,m_i}$ and a deadline, $D_i = \ell_c(m_i) + d_c$. It will then be inserted into Queue 1. Also, future message arrivals on this channel will be assigned logical times based on $\ell_c(m_i)$. This message cannot increase the maximum system time for any other messages, since the system times for these messages were computed by considering the maximum possible arrival rate for messages on channel M_c. Hence, a feasible priority-based schedule exists in this case, and consequently, a feasible EDD schedule also exists. ■

4 Flow Control and Buffer Management

Buffer space has to be reserved for real-time channels at the source, destination, and intermediate nodes in order to prevent buffer overruns and consequent loss of messages. During channel establishment, the user has to specify the burstiness in message generation, B_{max}, as part of the channel specification. This burstiness determines the buffer requirement at the source node of the channel. If $d_{c,s}$ is the worst-case delay guaranteed for channel M_c at the source node, the buffer requirement can be expressed as $(B^c_{max} + d_{c,s} \cdot R^c_{max})S^c_{max}$, where the $d_{c,s} \cdot R^c_{max}$ term accounts for new arrivals during the system (response) time of a message on the channel. It is necessary for the source node to provide this buffer space because the client can legitimately produce messages at this rate, and, if buffer space is not provided, messages could be lost due to non-availability of buffers. Also, the source node is responsible for making sure that message generation for M_c adheres to the channel specification. It can refuse to accept messages belonging to M_c when the actual generation rate exceeds the specified bounds.

Intermediate nodes also have to provide buffer space for M_c, but this need not depend upon B^c_{max} if a flow-control mechanism is employed between nodes. The buffer reservation scheme and the flow-control mechanism are intimately related, since flow control can be used to regulate the burstiness of message arrivals at intermediate nodes. The flow-control mechanism considered here operates by holding back some of the messages which arrive before their logical arrival time. If flow control were not used, messages could zip through lightly loaded nodes and back up at heavily loaded nodes, thereby causing buffer space problems at the heavily loaded nodes. An extreme case of flow control is when all messages which arrive early are held back, that is, a message is considered for transmission only when $\ell_c(message) \leq \text{current\_time}$. In this extreme case, an intermediate node, k, need only provide buffer space proportional to $d_{c,k} \cdot R^c_{max} \cdot S^c_{max}$ and rely on flow control to eliminate the burstiness. However, this strict regulation can result in unnecessary delays for messages when the network is lightly loaded. By allocating more buffer space, we make it possible for messages to go quickly through lightly loaded nodes.

The logical arrival time assigned to a message depends upon the burstiness in the arrival process. Hence, the bound on burstiness can also be interpreted as a *horizon* for the logical arrival time. Burstiness can be measured by the equation,

$$B = (\ell_c(\text{message}) - \text{current\_time})/I^c_{min}$$

which can be rewritten as

$$\ell_c(\text{message}) = \text{current\_time} + B \cdot I^c_{min}.$$

A bound on the burstiness, $B \leq B_m$, can therefore be interpreted as $\ell_c(\text{message}) \leq \text{current\_time} + B_m \cdot I^c_{min}$. Packets belonging to messages with logical arrival time greater than this horizon may be discarded by the node. In other words, the horizon, $B_m \cdot I^c_{min}$, determines the buffer requirement for a channel at an intermediate node.

A node can compute the horizon for a link, based on the available buffer memory, whenever a new channel is to be established with that link. This horizon is passed back to the upstream node, since that node is responsible for flow control on the link. The upstream node has to make sure that it transmits forward only those messages which have logical time, at that node, within the horizon (packets which belong to a single message all have the same logical time).

It is possible to use a different horizon for each channel, but this can create problems for the run-time scheduler because it is then forced to examine all the packets in Queue 3 individually to check their eligibility for transmission. Use of a single horizon means less flexibility in buffer management, but it is more suitable for run-time scheduling. The buffer space required for a channel is computed as follows.

Consider two adjacent nodes a and b, sharing a link (a, b) which has a horizon $H_{a,b}$. Consider a message p arriving at node b on channel M_c, using link (a, b). Since node a uses $H_{a,b}$ for flow control on link (a, b), we can deduce that the logical time for the message at node a must be $\ell_{c,a}(p) \leq \text{current\_time} + H_{a,b}$. The logical time assigned to the message at node b is $\ell_{c,b}(p) = \ell_{c,a}(p) + d_{c,a}$, where $d_{c,a}$ is the assigned worst-case delay for M_c at node a. Node b will not contain any messages belonging to M_c which have $\ell_{c,b}(m) < \text{current\_time} - d_{c,b}$, where $d_{c,b}$ is the worst-case delay assigned to the channel at this node. This is because messages on M_c are guaranteed to leave b within $d_{c,b}$ time units from their logical arrival time. Hence, node b can only contain messages belonging to M_c which have logical time in the range

$$\text{current\_time} - d_{c,b} \leq \ell_{c,b}(p) \leq \text{current\_time} + H_{a,b} + d_{c,a}.$$

The buffer space required to hold these packets is then given by:

$$\lceil (H_{a,b} + d_{c,a} + d_{c,b})/I^c_{min} \rceil \cdot S^c_{max}.$$

From this equation, it is clear that the minimum buffer space required for a channel is $S^c_{max} \cdot \lceil (d_{c,a} + d_{c,b})/I^c_{min} \rceil$. When the horizon H for a link is very large, the maximum buffer space required for a channel is bounded by $S^c_{max} \cdot (\lceil D_c/I^c_{min} \rceil + B^c_{max})$, where D_c is the end–to–end delay for the channel.

The overall buffer space requirement of a node is the sum of the buffer space required for all channels going through the node. When a new channel is being established, the node first tries to accommodate the new channel by computing the buffer space based on the existing horizon. If the available buffer space is less than the requirement, the node then reduces the horizon for that link. Reducing the horizon for a link has the effect of reducing the buffer space requirement of *all* channels using the link, so the amount of buffer space available increases.

5 Extensions for Long Messages

The channel establishment scheme, as presented, can handle messages of any size. However, the response time obtained for a long message can be substantial, and since the latency for message delivery is determined by the cumulative response time over the links in the route, it can grow very large. The underlying problem here is that store–and–forward delays grow with the size of the message. Long messages also pose problems for buffer management at intermediate nodes as these nodes have to reserve sufficient buffer space to hold the entire message. Since the number of channels going through an intermediate node can be large, the buffer space requirement of transit messages is an important issue.

Consider a request for a channel $M_m = (C_m, p_m, D_m)$, where D_m is the end–to–end delay for M_m, which has to traverse L_m links. In order to satisfy the end–to–end delay, the sum of the worst-case delays on the L_m links should be no greater than D_m. Hence, *on the average,* the worst-case delay at a link should be within D_m/L_m. It can be seen that as L_m increases, the delay available at each link reduces and it may not be possible to satisfy the delay requirement. Also, the buffer space required at each link for M_m depends upon C_m (and on the burstiness parameter).

The delay and buffer problems of long messages can be alleviated by splitting these messages into *packet groups* and "pipelining" the transmission of these packet groups[2]. A message on channel M_m can be split into N_m packet groups, each with size $C_{grp} = C_m/N_m$ and delay d_{grp}. The transmission of these packet groups will be in sequence, as shown in Figure 7. This figure shows a message split into 2 packet groups, C^1_{grp} and C^2_{grp}, and transmitted over three links. Transmission of each packet group has to complete within an interval of length d_{grp}. Here d_{grp} represents the delay in any stage of the pipeline, and it has to exceed (or equal) the maximum response time for the packet group over the links in the route. The pipelining mechanism enables us to bring out the overlap between the transmission of packets belonging to a message of M_m on different links of the route. The end–to–end delay for a message on M_m is then $d_{grp} * (N_m + L_m - 1)$ ($= 4d_{grp}$ in the example). Hence, the constraint that d_{grp} has to satisfy is:

$$d_{grp} \leq D_m/(N_m + L_m - 1). \tag{5.1}$$

Selection of N_m: When packet groups are used, a larger value for N_m is preferable from the standpoint of the delay computation. The ratio C_{grp}/d_{grp} defines the fraction of time required to process the packet group to the packet period, and it is desirable to make this small. This ratio, when expressed in terms of the parameters of M_m, is $(C_m/D_m)\cdot(N_m + L_m - 1/N_m)$, a monotone decreasing function of N_m. Also, the buffering problem is reduced because intermediate nodes now have to provide buffer space based on the size of the packet group instead of the message size. To increase flexibility in buffer management, it appears that we should increase the number

[2]We use the term packet group to distinguish this entity from a physical packet. A packet group can contain one or more physical packets.

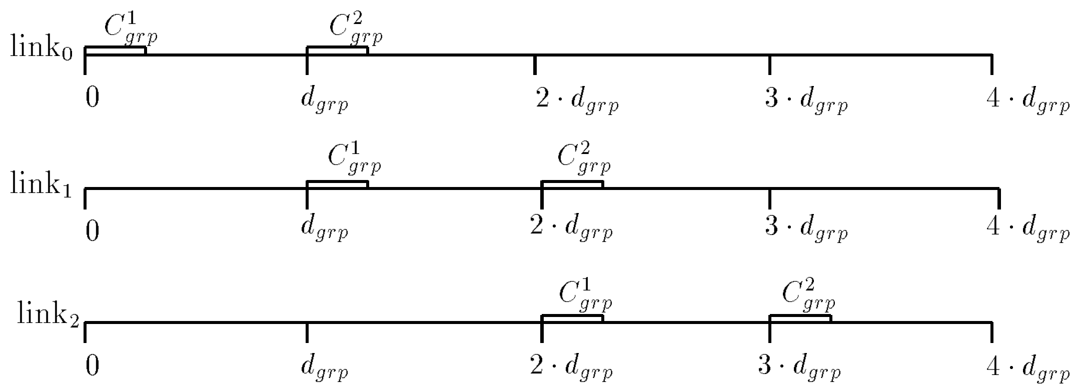

Figure 7: Packet group illustration.

of packet groups, N_m, as much as possible, preferably to have only one packet per packet group.

In this pipelining scheme, it is necessary that all packet groups of a message should be transmitted from a link before the arrival of the next message on the channel. This imposes a restriction on the possible values for the number of packet groups.

$$d_{grp} \cdot N_m \leq p_m \tag{5.2}$$

Using Eqs. (5.1) and (5.2) we get

$$(D_m/(N_m + L_m - 1)) \cdot N_m \leq p_m \tag{5.3}$$

This equation can also be rewritten as a constraint on N_m, whose solution depends upon the relative values of D_m, p_m, and L_m:

$$1/L_m \leq N_m/(N_m + L_m - 1) \leq p_m/D_m. \tag{5.4}$$

There are three cases to be considered:

$D_m < p_m$ **:** In this case, N_m can take any value. However, for large N_m, d_{grp} is small and this implies that the priority assigned to the channel is high. This can interfere with the resource requirements of other channels, and result in the rejection of subsequent channel establishment requests.

$p_m < D_m \leq Lp_m$ **:** The optimal choice for N_m is the largest integer which satisfies Eq. (5.3). For example, if $D_m = 2p_m$ in the example of Figure 7, the optimal choice for N_m is 2.

1. Select a source–destination route for the channel. This determines L_m, the number of links on the route.
2. Select N_m, the number of packet groups.
3. Compute the worst-case delay for the message on each link of the route. The desired worst-case delay is set to $d_m = D_m/(N_m + L_m - 1)$.
4. Check the buffer requirement and compute the horizons for the different links.
5. Check whether the channel can be accommodated (Figure 9) for each link on the route. This is the channel establishment test.
6. If the channel can be established, assign d_m as the delay for each link on the route.

Figure 8: **Channel establishment for packet groups.**

$D_m > Lp_m$ **:** It can be seen that Eq. (5.4) has no solution when $D_m > L \cdot p_m$. However, the end–to–end delay requirements are not very stringent in this case. Although pipelining cannot be used, we still have message level parallelism for transmission on different links.

In practice, the choice of N_m is also affected by the size of the resulting packet group and the group delay. d_{grp} cannot be made very small because it has to include the transmission time of at least one packet, since packet transmission cannot be interrupted.

5.1 Channel Establishment

When a channel creation request is handled, we have to decide whether this request should be treated as a regular channel or a channel with packet groups. This decision is based on the parameters of the channel, D_m and p_m, and the number of links on the route to the destination. The channel establishment procedure has to be modified when packet groups are used, as shown in Figure 8. The decision on whether to consider the request as a channel with packet groups is taken in Step 2. The worst case delay for a link is again computed using procedure D_Order, but in this case, there is a target worst-case delay $d_m = D_m/(N_m + L_m - 1)$ for the packet group on each link. The channel establishment check is that each link should be able to satisfy this target worst-case delay, without violating the worst-case delay of existing channels.

Response Time Computation: The procedure D_Order also has to be modified to handle packet groups (see Figure 9). Since the channel to be established now has a target worst-case

1. Arrange the existing channels in ascending order of their associated delay d_i, where the delay is for a packet group of the channel.
2. Insert the new packet group into this order based on the required worst-case delay d_m.
3. Compute the new worst-case response times r'_i for all channels based on this priority assignment.
4. Check whether for all channels i, $r'_i \leq d_i$. If not, the channel cannot be established.

Figure 9: **Extensions to Procedure D_Order for packet groups.**

delay specified, it has to be assigned a priority based on this target d_m. This priority assignment can increase the worst-case response time, r'_i, for existing channels, and may cause some of them to exceed the assigned worst-case delay, d_i. The new channel can be accepted only if $r'_i \leq d_i$ for all channels.

The response time computation scheme used in procedure D_Order now has to accommodate packet groups. Consider an existing channel which has response time r such that $i \cdot d_m < r < (i+1)d_m$, for some $i < N_m$. This channel can consider M_m to be a channel with period d_m and a message length equal to the size of the packet group. However, channels which have $N_m \cdot d_m < r$ will treat M_m as a regular channel with length C_m and inter-arrival time p_m.

Run-time Scheduling: Packets belonging to a packet group are considered as part of a single unit, and use the same logical arrival time. The logical arrival time of a packet group is determined on the basis of the logical arrival time of the message, and the sequence number of the packet group. For example, the logical arrival time for a packet belonging to the ith packet group of the message m_j is:

$$\ell t(packet) = i \cdot d_m + \ell(m_j)$$

where $\ell(m_j)$ is defined as before for a regular channel. This computation has to be done only at the source node. At intermediate nodes, the logical arrival time is computed simply from the logical arrival time at the previous link. Here again, we can see how the use of a global time-base simplifies the design. Scheduling for the packet is handled as before based on this logical time and the horizon for the link.

Buffer Requirement: The intermediate nodes should provide buffer space sufficient to hold at least two packet groups. The buffer requirement is computed based on the horizon, as in

Section 4, but with some small changes. The channel is considered to have an inter-arrival time of d_m and a message size equal to the size of the packet group, C_m/N_m.

6 Related Work

The concept of a unidirectional real-time channel was developed by the researchers of the DASH project. The real-time channel considered in this paper is similar to the deterministic real-time channel of [7]. That paper also considers two other types of channels: statistical, where messages are given a guarantee of successful delivery with a certain probability, and best-effort (with no guarantees). We do not consider statistical channels mainly because real-time applications require hard guarantees. Comer and Yavatkar have developed an abstraction called *flows* [19], which is similar to a unidirectional channel, but they do not give guarantees for the delivery time of messages. Flows can be identified with best-effort channels.

The parameters used for the description of the message arrival process are based on the linear bounded arrival process model of Cruz [12], with small modifications. Although the outline of our channel establishment scheme is similar to the one presented in [7], it is different in several respects. We have examined the problem of guarantee computation in depth, and developed a scheme for computing worst-case response times based on priorities. This corresponds to the delay bound test of [7]. It can be shown that the delay bound test of [7] is a special case of the priority-based response time computation scheme presented here. Moreover, we have presented an integrated solution to the problem of buffer reservation and flow control, based on the *horizon* model. We have also shown that channels with long messages require special consideration, and presented a scheme to accommodate them.

The problem of time-constrained communication in a multi-hop network has also been addressed by Cidon *et al.* in the PARIS system [20, 21]. Their approach is based on providing limited buffer space in the switching nodes and using FIFO scheduling to obtain an upper bound on the network delay in each node. Limited buffering can result in packet loss, and so their efforts are aimed at maximizing throughput, while keeping the probability of packet loss below a certain level. Another approach to this problem is the "stop–and–go" queueing framework proposed by Golestani [22, 23]. In this approach, a *framing* technique is used at all switching nodes to eliminate packet loss and to achieve a guaranteed end–to–end delay. However, the end–to–end delay is tied to the sizes of the frames, thus reducing the flexibility of satisfying different delay requirements. It appears that this approach is appropriate mainly when the required end–to–end delay is large compared to the periodicity of the message arrivals, as in voice transmission.

7 Conclusions

In this paper, we have identified and solved problems related to the establishment and management of real-time communication channels. We have presented algorithms for computing the worst-case delay for messages, and for scheduling these messages. The computation was based on priority scheduling, and it was shown to be optimal under certain conditions. We also presented mechanisms for buffer allocation and flow control suitable for real-time channels, which would preserve the guarantees on delivery time. Using the ideas presented here, we are currently building a prototype communication subsystem for a distributed real-time system, called HARTS [10, 24], which is being developed at the Real-Time Computing Laboratory, The University of Michigan.

References

[1] W. Zhao and K. Ramamritham, "Virtual time CSMA protocols for hard real-time communication," *IEEE Transactions on Software Engineering*, vol. 13, pp. 938–952, Aug. 1987.

[2] J. F. Kurose, M. Schwartz, and Y. Yemini, "Controlling window protocols for time-constrained communication in multiple access networks," *IEEE Trans. Communications*, vol. 36, pp. 41–49, Jan. 1988.

[3] W. Zhao, J. A. Stankovic, and K. Ramamritham, "A multi-access window protocol for transmission of time constrained messages," in *Proc. 8th International Conference on Distributed Computing Systems*, pp. 384–392, June 1988.

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

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

[6] D. Ferrari, "Guaranteeing performance for real-time communication in wide-area networks," Tech. Rep. UCB/CSD 89/485, UCB Computer Science Division, EECS, Berkeley, CA, Jan. 1989.

[7] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal on Selected Areas in Communications*, vol. SAC-8, pp. 368–379, Apr. 1990.

[8] D. P. Anderson, "A software architecture for network communication," in *Proc. 8th International Conference on Distributed Computing Systems*, pp. 376–383, June 1988.

[9] D. P. Anderson and D. Ferrari, "The DASH project: An overview," Tech. Rep. 84/405, UCB Computer Science Division, EECS, Berkeley, CA, Feb. 1988.

[10] J. W. Dolter, P. Ramanathan, and K. G. Shin, "A microprogrammable VLSI routing controller for HARTS," in *Proc. IEEE Int'l. Conf. on Computer Design: VLSI in Computers*, pp. 160–163, IEEE, Oct. 1989.

[11] P. Ramanathan, D. D. Kandlur, and K. G. Shin, "Hardware assisted software clock synchronization for homogeneous distributed systems," *IEEE Trans. Computers*, vol. C-39, pp. 514–524, Apr. 1990.

[12] R. L. Cruz, *A Calculus for Network Delay and a Note on Topologies of Interconnection Networks*. PhD thesis, University of Illinois at Urbana-Champaign, July 1987. available as technical report UILU–ENG–87–2246.

[13] D. P. Anderson, S. Y. Tzou, R. Wahbe, R. Govindan, and M. Andrews, "Support for continuous media in the DASH system," in *Proc. 10th International Conference on Distributed Computing Systems*, pp. 54–61, IEEE, May 1990.

[14] D. P. Anderson, R. G. Herrtwich, and C. Schaefer, "SRP: A resource reservation protocol for guaranteed performance communication in the internet," Technical Report TR-90-006, International Computer Science Institute, Berkeley, Feb. 1990.

[15] D. D. Kandlur and K. G. Shin, "Traffic routing for networks with virtual cut-through capability," in *Proc. 10th International Conference on Distributed Computing Systems*, pp. 398–405, IEEE, May 1990.

[16] M. L. Dertouzos, "Control robotics: The procedural control of physical processes," in *Proceedings IFIP Congress*, pp. 807–813, 1974.

[17] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, pp. 46–61, Jan. 1973.

[18] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, Dec. 1982.

[19] D. E. Comer and R. Yavatkar, "FLOWS: Performance guarantees in best effort delivery systems," Technical Report CSD-TR-791, Computer Science Department, Purdue University, West Lafayette, IN 47907, July 1988.

[20] I. Cidon and I. S. Gopal, "PARIS: An approach to integrated high-speed private networks," *Intl. Journal of Digital and Analog Cabled Systems*, vol. 1, pp. 77–86, Apr. 1988.

[21] I. Cidon, I. Gopal, G. Grover, and M. Sidi, "Real-time packet switching: A performance analysis," *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 1576–1586, Dec. 1988.

[22] S. J. Golestani, "Congestion-free transmission of real-time traffic in packet networks," in *Proc. INFOCOM*, pp. 527–536, IEEE, June 1990.

[23] S. J. Golestani, "A stop-and-go queueing framework for congestion management," in *Proc. SIGCOMM*, pp. 8–18, ACM, Sept. 1990.

[24] M.-S. Chen, K. G. Shin, and D. D. Kandlur, "Addressing, routing, and broadcasting in hexagonal mesh multiprocessors," *IEEE Trans. Computers*, vol. C-39, pp. 10–18, Jan. 1990.

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

A Schedulability Analysis

Consider a set of channels $\{M_i = (C_i, d_i, p_i), i = 1, \ldots, m\}$ which share a common link ℓ, where C_i is the maximum service time on ℓ for any message on channel M_i, d_i is the maximum permissible delay for messages belonging to M_i at this link, and p_i is the inter-arrival time. Assume that the channel set is ordered by priority with M_1 being the channel with the highest priority. It can be shown that the worst-case delay for a message occurs when its arrival time coincides with the arrival time of all other messages of equal or higher priority. We can denote this critical instant as time $t = 0$. The transmission of a message can be delayed by the instances of higher priority messages present in the system at $t = 0$ and by subsequent arrivals of new instances of these messages. The arrival times of these instances are given by the set S_i, and the system time requirement of this message and of the higher priority messages is given by $W_i(t)$, as shown below (see [25]).

$$S_i = \{d_i\} \bigcup \{kp_j \mid j = 1, 2, \ldots, i-1; k = 1, 2, \ldots, \lfloor (d_i/p_j) \rfloor\}$$

$$W_i(t) = \sum_{j=1}^{i-1} C_j \cdot \lceil t/p_j \rceil + C_i \tag{A.1}$$

A channel M_i is schedulable (i.e., its delay bound will always be satisfied) *if and only if* $W_i(t) \leq t$, for some $t \in S_i$. The worst-case response time for messages belonging to M_i is the smallest value of t such that $W_i(t) = t$. The set of channels is said to be schedulable if and only if each channel in the set is schedulable.

These results hold under the conditions that preemptive scheduling is employed and a message can be aborted at any stage in its execution without any loss. In our model, messages are split into packets, and packet transmission cannot be interrupted. The schedulability conditions have to be modified to account for the new worst-case conditions. The worst-case response time for a message occurs when all messages of equal and higher priority arrive at the same time and a packet belonging to a lower priority message is being transmitted at that time. The system time requirement in this case is:

$$W_i'(t) = C_{pkt} + \sum_{j=1}^{i-1} C_j' \cdot \lceil t/p_j \rceil + C_i', \tag{A.2}$$

where C_{pkt} is the service time for the maximum size packet, and C_j' is the service time for a message on M_j, including the overheads of packetization.