A Distributed Table-Driven Route Selection Scheme for Establishing Real-Time Video Channels

Chih-Che Chou and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
Email: {ccchou,kgshin}@eecs.umich.edu; 313-763-0391 (voice); 313-763-4617 (Fax)

September 30, 1994

ABSTRACT

To guarantee the delivery of real-time messages before their deadline, a real-time channel or connection must be established before the transmission of any real-time messages. During this channel-establishment phase, one must first select a route between the source and destination of this channel and then reserve sufficient resources along this route so that the worst-case end-to-end delay over the selected route may not exceed the user-specified delay bound.

We propose a table-driven distributed route-selection scheme that is guaranteed to find a "qualified" route, if any, that meets the performance requirement of the requested channel without compromising any of the existing guarantees. The proposed scheme uses the Bellman-Ford shortest path algorithm to build real-time delay tables, and hence, can solve the route-selection problem by a simple table look-up. Several examples are presented to demonstrate the effectiveness of the proposed distributed route-selection scheme.

Index Terms — real-time communication, real-time channel, video channels, routing tables, end-to-end communication delay bounds.

ICDCS Category — **Communication Protocols**

The work described in this paper was supported in part by the Office of Naval Research under Grant N00014-J-92-1080 and the National Science Foundation under Grant MIP-9203895. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of the funding agencies.

1 Introduction

An increasing number of applications such as interactive video and computer-integrated manufacturing require real-time networking services. Among the several real-time communication protocols proposed thus far to meet this requirement, the communication abstraction called the "real-time channel" [8] has received considerable attention due to its conceptual simplicity. A real-time channel is a unidirectional virtual circuit which, once established, is guaranteed to meet user-specified performance requirements as long as the user does not violate his *a priori* specified traffic-generation characteristics [8].

There are two distinct phases with the realization of a real-time channel: off-line channel establishment and run-time message scheduling. During the channel-establishment phase the system has to select a route between the source and destination of the channel along which sufficient resources can be reserved to meet the user-specified delay and buffer requirements. Although several channel-establishment schemes have been proposed in the literature [4–6, 8–10, 16], very few of them have addressed explicitly (and rigorously) the issue of selecting a route between the source and destination of a channel, despite its importance to the success/failure of channel establishment.

Since the number of possible routes between two communicating peers in a multi-hop network could be large, selecting a route for each real-time channel is potentially a time-consuming task. It is therefore very important to develop an efficient scheme that is guaranteed to find a "qualified" route, if any, for each requested real-time channel. If the worst-case anticipated traffic over a real-time channel is given (typically in terms of the minimum message inter-arrival time and the maximum message size), a "qualified" route for this real-time channel is defined to be the one that can meet the user-specified end-to-end delay requirement without compromising any of the existing guarantees. (There are other performance requirements like jitter bounds and message loss rate to meet, but for clarity we will focus only on the end-to-end delay requirement.) The service provider (the network operating system in our case) must also be able to reject a channel-establishment request as soon as possible if no qualified routes are available for the requested channel.

There are basically two approaches to the route-selection problem: centralized or distributed. Most existing channel-establishment schemes are centralized [4, 5, 9, 10, 16]. They simply assume the existence of a global network manager which maintains the information about all the established real-time channels, the topology and resource distribution & commitment of the underlying network, and can thus select an appropriate route for each real-time channel requested. In such a centralized scheme, all of real-time channel-establishment requests require the network manager's approval. That is, each channel-establishment re-

quest is sent, along with its traffic-generation characteristics and user-specified performance requirement, to the network manager, which then selects a qualified route and reserves resources along the selected route. The network manager also informs all intermediate nodes on this route of the establishment of the new channel and the information necessary for run-time scheduling of the messages of this channel. Although using a centralized scheme one can devise efficient algorithms for the network manager to select qualified routes, there are two serious problems with this scheme. First, the network manager is likely to be a performance bottleneck, since it must handle all channel establishment and disconnection requests. Second, the system is susceptible to single-point failures of the network manager, since without the network manager real-time channels cannot be added or deleted.

In contrast with the centralized approach, the distributed route-selection approach can avoid performance and reliability bottlenecks. However, it generally suffers the following problem. Since there could be many possible routes between two communicating peers, it may be too time-consuming to search for all possible routes and perform an admission test on each candidate route during the channel-establishment phase. On the other hand, if we only test a small number of routes, we may not find a qualified route even if there exists one. That is, the distributed approach may reject a channel-establishment request even when it would have been accepted if more routes had been checked.

In a recent paper [15], we have proposed a *generic* distributed route-selection scheme which is guaranteed to find a qualified route, if any, for each real-time channel-establishment request. Basically, the scheme uses the algorithm proposed in [10, 11] to estimate the minimum worst-case response time (MWRT) of a real-time channel over each link of a route under consideration. By using MWRTs, we then applied the Bellman-Ford algorithm [2, 19] to solve the distributed route-selection problem. However, this scheme suffers the problem of over-estimating link delays. That is, when there are multiple *simultaneous* channel-establishment requests, the link delays may be over-estimated as a result of treating the pending channels as if they had already been established, thus sometimes incorrectly rejecting channel-establishment requests; they could have been accepted if link delays had not been over-estimated. Another problem with the scheme in [15] is that it does not take advantage of the underlying application features, because it was intended for *general* real-time applications. In other words, if we only have to support real-time channels with limited, yet important, types — like interactive video — of traffic-generation behaviors and user's performance requirements, we can improve the efficiency and performance of the route-selection scheme significantly, e.g., shorter channel-establishment delay and higher channel acceptance ratio.

In this paper, we propose a distributed table-driven route-selection scheme which takes

advantage of the highly-periodic nature of interactive video and maintains a *real-time delay table* (or simply *delay table*) at each node so that selecting a route for a real-time video channel may be done with a simple table look-up. Considering the growing importance of interactive video, narrowing the scheme's applicability to interactive video for better performance is an important choice that should be made available to the system designers.

The paper is organized as follows. Section 2 states the problem of finding a qualified route for each real-time video channel. Our proposed solution to this problem is presented in Section 3. In Section 4 we demonstrate via examples the effectiveness of the proposed solution. The paper concludes with Section 5.

2 Problem Statement

In view of the reliability and performance bottlenecks of centralized route selection, the distributed approach is a more promising alternative. In order to choose a qualified route for a new real-time channel requested, the system has to perform the admission test on each route to see if there are sufficient resources along the route to meet the user-specified end-to-end delay requirement for this channel. (Although there may be other performance requirements to be met, for clarity of presentation we will focus on the user-specified end-to-end delay requirement.)

There could be a large number of routes between two communicating peers, making it difficult/time-consuming to choose a qualified route among all possible routes between the source and destination of each requested channel. Conventionally, the system does not start the route-selection process until a channel-establishment request (with the performance requirement and traffic-behavior specification) is received. The system usually incurs a long channel-establishment delay or very high operational overhead. There are two simple-minded approaches to the distributed route-selection problem:

1. Sequential search of all possible routes one by one, or K routes at a time.
2. Parallel search of all possible routes, i.e., sending multiple copies of an establishment request through all possible routes, "conditionally" reserving resources and performing admission tests on all of them.

The second approach is practically infeasible due to its excessive operational overhead. The first approach, on the other hand, could be very time-consuming for the complete search of all possible routes, and its operational overhead is proportional to K.

Considering the advantages and disadvantages of these two approaches, we developed

in [15] an efficient scheme for selecting a route for each channel-establishment request. Although this scheme is guaranteed to find a qualified route, if any, for a *single* channel-establishment request, it cannot guarantee the qualified routes to be found for multiple (near) *simultaneous* channel-establishment requests due to its reliance on the over-estimation of link delays. Since this scheme is intended for channels with *general* performance requirements and traffic patterns, its performance could be improved significantly if real-time traffic is limited to certain types. Specifically, we will in this paper consider the real-time traffic of interactive video applications. Interactive video applications usually generate frames at some fixed rate[1] and resolution which are both specified according to industry standards. For example, 30 frames per second is the frame rate for live interactive video and the MPEG Video Simulation Model Three (SM3) suggests 352 by 288 pixels per frame for achieving video tape quality [12]. Note that a standardized resolution implies a standardized maximum-frame size. Since video applications of our interest require only a small set of combinations of frame-generation rates and maximum-frame sizes, by exchanging and maintaining real-time traffic information among the nodes, the system may be able to prepare for channel establishment even *before* receiving a establishment request. Under this setting, we will develop a scheme which builds and maintains a delay table on each node so that the route-selection problem can be solved by a simple table look-up at the source node.

3 The Proposed Solution Approach

We first describe the environment and the assumption under which our distributed route-selection scheme will be developed. The underlying network is an arbitrary point-to-point network. As in [1, 7, 10, 11, 15, 18], the generation of real-time messages is assumed to be governed by a linear-bounded model that is characterized by three parameters: maximum message size S_{max} (bytes), maximum message rate R_{max} (messages/second), and maximum burst size B_{max} (messages). In the linear bounded model, there are two restrictions:

- The number of messages generated in any time interval of length t does not exceed $B_{max} + tR_{max}$.
- The length of each message is bounded by S_{max}.

Based on this message arrival model, the authors of [10, 11] proposed a scheme to estimate the worst-case delay on each link and a run-time scheduling algorithm for real-time messages.

[1] allowing jitters

By adding the worst-case delays of all links that a channel runs through, one can calculate the worst-case end-to-end delivery delay. This end-to-end delay is then compared against the user-specified end-to-end delay bound for the requested channel and the system can decide whether to accept/reject the corresponding real-time channel-establishment request. Note that these schemes have been developed under the assumption that a proper route for the requested channel was already available.

In addition to the linear bounded model, we further assume that the number of possible combinations of frame-generation rates and maximum-frame sizes is small, since we are only interested in standardized interactive video applications. Based on the link delay calculated with the delay-estimation method in [10, 11] and the above assumptions, we will develop a scheme which builds real-time channel delay tables on each node so that a qualified route may be found with a simple table look-up.

3.1 Link-Delay Estimation

Since real-time messages are given priority over non real-time ones, we will ignore the effects of non real-time traffic in the rest of the paper unless stated otherwise. We will thus assess the delay of a link based only on real-time traffic. Since the algorithm in [10, 11] will be used to estimate link delays, we will briefly introduce it first.

The goal of the algorithm in [10, 11] is to compute the minimum worst-case delay on a link of each candidate route for a new real-time channel to be added without compromising the performance guarantee of any of the existing channels on the link. Let $\{M_i = (C_i, p_i, d_i), i = 1, \ldots, k\}$ be the set of k existing channels on a link, where C_i is the maximum time required to transmit a message of channel M_i on the link, p_i is the minimum message inter-arrival time in M_i, and d_i the maximum permissible delay assigned to M_i on this link, or *link deadline*. Note that the inequality $d_i \leq p_i$ must hold for the algorithm in [10, 11] to work correctly. Given a new channel $M_{k+1} = (C_{k+1}, p_{k+1})$ to be established, the authors of [10, 11] proposed an algorithm for computing the minimum worst-case response time (MWRT), r_{k+1}, on each link of M_{k+1}'s route without compromising the performance guarantees of other existing channels. The algorithm statically assigns priority to each real-time channel (in ascending order of d values) to calculate the MWRT for M_{k+1}, but uses an Earliest-Due-Date (EDD) algorithm for run-time scheduling. The algorithm can compute the MWRT for a new channel on link ℓ based on the traffic-generation characteristics (C and p) of the channel, when C (maximum service time for a message), p (minimum message inter-arrival time) and d (maximum permissible delay over link ℓ) for all existing channels are available.

The method in [10, 11] does *not* include those channels pending for final confirmation in the calculation of MWRT for the new channel-establishment request, but we will treat pending channels differently in two situations. First, during the channel-establishment phase, we will include the load of pending channels in the calculation of MWRTs as if they had already been established. However, the load of pending channels will *not* be included in the real-time delay tables, i.e., we do not include pending channels in the calculation of MWRTs which are used to build real-time delay tables.

Including pending channels in the calculation of MWRTs can simplify the channel-establishment phase, since the MWRTs remain valid when the confirmation message travels back from the destination to the source. Otherwise, the MWRT for a new channel may change due to the confirmation of other pending channels which share one or more links with this channel, and thus, we have to check this possibility at every intermediate node the confirmation message visits en route to the source node. On the other hand, inclusion of these pending channels in the link-delay estimation will sometimes make the MWRT larger than it actually would be if some of them are rejected later. This over-estimation of MWRT may result in incorrect rejections of channel-establishment requests. Fortunately, the over-estimation problem occurs only when two requests initiate at about the same time and their least-MWRT paths share at least one link. As will be seen later, because the proposed scheme checks only one least-MWRT route (known thus far) at a time instead of checking all possible routes in parallel (the scheme in [15] does this), the over-estimation of MWRT is much less likely to occur. Thus, the incorrect rejection decisions due to the over-estimation of MWRT are likely to be made only when there is a very high percentage of real-time traffic so that the over-estimation of MWRT over those shared links may make the end-to-end delay larger than the application-required latency. Since a good system design should also anticipate the existence of a substantial percentage of non real-time traffic, the over-estimation problem is usually not serious. In order to avoid any possible confusion, "existing channels" will henceforth mean *both* established and pending channels. Note that different real-time channels have different traffic-generation patterns, and hence, each of them is associated with a different MWRT, i.e., different channels may have different MWRTs over the same link. Determination of each channel's MWRT on a link will be referred to as *link-delay estimation*. Given below is a simple example illustrating the link-delay estimation.

Example 1: Suppose the link from node A to node B has a bandwidth of 50 million bits per second (Mbps), and Table 1 shows the five established real-time channels which use this link ($A \rightarrow B$). Note that in order to be consistent with the way channel priorities are assigned for the link-delay estimation, the entries in Table 1 are placed in ascending order of d values, i.e., the highest priority is given to the channel with the least permissible delay on

ID	C (ms)	p (ms)	d (ms)
1	2.0	10	5.5
2	1.0	20	8
3	0.6	20	16.2
4	2.0	20	20
5	2.0	40	20.5

Table 1: Example 1

this link. However, this priority assignment is used only for calculating the MWRT; a multi-class EDD algorithm is used for the run-time scheduling of message transmissions. (The optimality of EDD in meeting deadlines legitimates the off-line calculation of MWRT with fixed-priority scheduling followed by the on-line EDD scheduling of message transmissions.) Using this environment, we will demonstrate the computation of MWRTs for a real-time channel.

Suppose the traffic of the requested channel is specified by $S_{max} = 200$ Kb and $p = 20$ ms. Then $C = 200/50 = 4$ ms. Note that the end-to-end delay requirement is not needed in this example since we only demonstrate how the link delay estimation procedure works. To compute a channel's MWRT on a link, we need to assign the highest possible priority to this channel without violating the link deadlines (d in Table 1) of other existing channels. We start with assigning the requested channel top priority, then check if any of the existing channels' link deadlines will be violated. By giving the newly-requested channel top priority on this link, the worst-case delay of channel 1 will be $4 \times \lceil \frac{5.5}{20} \rceil + 2 = 6 > 5.5$. So, we lower the priority of this new request below channel 1 but above channel 2. With this new priority assignment, no existing guarantees will be violated, because the following schedulability test holds for any channel i whose priority is lower than the requested channel [10, 11].

$$\exists t \in B_i = \{d_i\} \cup \{kp_j : \ j \in A_i, 0 < k \leq \lfloor \frac{d_i}{p_j} \rfloor\},$$

$$W_i(t) = \sum_{j \in A_i} C_j \cdot \lceil \frac{t}{p_j} \rceil + C_i \leq t,$$

where A_i is the set of channels (including the requested channel) whose priority is higher than that of channel i. Given below is an 'acceptable' set of t and $W_i(t)$ for channels on this link:

- channel 2, $t = d_2 = 8$, $W_2(8) = 7 < 8$.
- channel 3, $t = d_3 = 16.2$, $W_3(16.2) = 9.6 < 16.2$.
- channel 4, $t = d_4 = 20$, $W_4(20) = 11.6 < 20$.

- channel 5, $t = p_5 = 20$. $W_5(20) = 13.6 < 20.5$.

(Note that the above numbers for channels 3–5 are *not* worst-case response times, but merely 'acceptable' numbers.) As a result, the priority of the requested channel will be placed between channel 1 and channel 2. So, the MWRT of the requested channel on this link is equal to the smallest t such that $W_{new}(t) = t$, i.e., the smallest t such that $2 \times \lceil \frac{t}{10} \rceil + 4 = t$. This channel's MWRT on this link is therefore 6, i.e., the delay of this link is 6 ms. 2

3.2 Building Real-Time Delay Tables

Based on the above definition of link delay, we can apply the Bellman-Ford algorithm [2, 19] and a loop-free version of the APARNET's previous routing strategy (APRS) [3, 13, 14, 17] to build real-time delay tables on each node. As mentioned earlier, the MWRT used to construct real-time delay tables does not include the load of pending channels for two reasons:

- The maximum permissible delay for a link after a final confirmation is likely to be greater than the MWRT computed during the resource-reservation phase. (We will discuss more on this in Section 3.3.)
- The time between making resource reservation and receiving a final confirmation is usually small, e.g., it could be the time needed for the round-trip from a node on the route under test to the destination.

Thus, if we want to include the load of pending channels in the real-time delay tables, the table entries may have to be modified twice in a short period of time.

When only a small set of standardized combinations of frame-generation rates and maximum-frame sizes needs to be considered for real-time video channels, each node in the network can build a loop-free table based on the MWRTs computed according to a pair of maximum-frame size (S_{max} or C) and frame-generation rate (p). All real-time channels that can be specified by the same pair (S_{max}, p) are said to be in the same *class*. A node will compute the MWRT on each of its outgoing links as the minimum feasible delay of the corresponding link for *each class* of real-time channels. These MWRTs will be stored in a table, TM, which can be indexed by its neighbors' addresses and has only one field, r, for each class of channels, representing their MWRTs over the corresponding link by considering only those channels already established.

Since the loop-free version of APRS will be used to exchange delay information and maintain real-time delay tables, we will briefly describe the original APRS. In APRS, each node collects and maintains the information about the minimum delays to all the other nodes via each of its neighbors. Thus, for every destination–neighbor pair, the information is kept in a 3-tuple form (*destination*, *neighbor*, *delay*). Other information may be needed for the loop-free version of APRS, but we will not discuss this issue here. (See [3, 13, 14, 17] for a detailed account of this.) These 3-tuples are divided into groups based on the destination node. Within a group, they are listed in the ascending order of delay. The first entry of each group (the minimum-delay entry to the corresponding destination node) is then used to build a routing table.

Each node periodically exchanges a routing message with its neighbors which contains the node's current routing table. Again, recall that the loop-free version needs to exchange some additional information, and the routing table exchanged may be slightly different from that of the original APRS. After receiving routing messages from its neighbors, each node will update its own routing table based on the information carried in the routing messages and the status of its own outgoing links. As we shall see, the real-time delay table is built in the same way as the routing tables except for the following two differences.

- Only real-time traffic is considered when building real-time delay tables. (APRS's routing tables were built by considering *all* traffic.)
- Real-time delay tables are updated only when a new real-time channel is established or an existing real-time channel is closed. Thus, a node sends "routing" messages to its neighbors only when its set of real-time channels changes. These "routing" messages are called *real-time routing* messages.

Using an example, we will show how to build real-time delay tables.

Example 2: The class of real-time channels under consideration is specified by $S_{max} = 100$ Kbps and $p = 33$ ms. Fig. 1 shows the network used for this example. Labeled on each link is a number which represents its transmission speed. Since initially there are no existing real-time channels, if a channel-establishment request of this class is received, the highest priority will be given to the requested channel. Therefore, the MWRT of this class for each link is equal to the maximum service time (C) of the class and can be computed as: link $N1 \leftrightarrow N2$: 100/100=1 ms; link $N1 \leftrightarrow N3$: 100/50=2 ms; link $N2 \leftrightarrow N4$: 100/50=2 ms; link $N3 \leftrightarrow N4$: 100/20=5 ms; link $N3 \leftrightarrow N5$: 100/10=10 ms; and link $N4 \leftrightarrow N5$: 100/50=2 ms.

Table 2 shows the real-time delay tables for all 5 nodes in the very beginning. Initially, a node can reach only its neighbors since information about the other nodes is not yet available

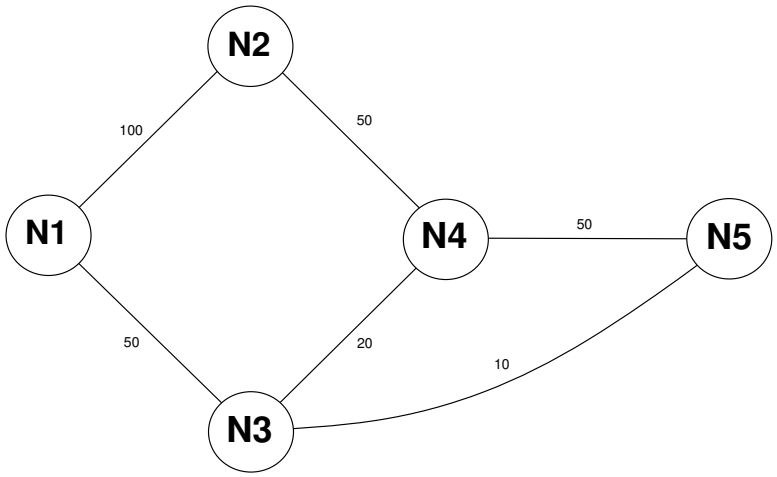

Figure 1: Example 2

N1's table			N2's table			N3's table			N4's table			N5's table		
D	N	d	D	N	d	D	N	d	D	N	d	D	N	d
2	2	1	1	1	1	1	1	2	2	2	2	3	3	10
2	3	∞	1	4	∞	1	4	∞	2	3	∞	3	4	∞
3	3	2	4	4	2	1	5	∞	2	5	∞	4	4	2
3	2	∞	4	1	∞	4	4	5	3	3	5	4	3	∞
						4	1	∞	3	2	∞			
						4	5	∞	3	5	∞			
						5	5	10	5	5	2			
						5	1	∞	5	2	∞			
						5	4	∞	5	3	∞			

Table 2: Initial real-time delay tables, where D = Destination, N = Neighbor, d = delay.

to the node. The label ∞ in an entry represents the case when either the destination cannot be reached via the corresponding neighbor or the path via this neighbor is not loop-free.

The least-MWRT path to each destination known so far is used to construct a real-time routing message. Each entry of the message is a 2-tuple, $(destination, delay)$, where *destination* is not the neighbor to which this message will be sent. For example, the message from N3 to N4 will contain the two entries: $(N1, 2)$ and $(N5, 10)$.

Since the real-time routing messages are not sent periodically (i.e., the updating procedures of real-time delay tables are not synchronized among nodes), it is not certain what the "next" state of the real-time delay tables will be. However, the "steady-state" real-time delay tables depend only on the currently-anticipated real-time traffic of the established real-time channels. Thus, after all nodes stop sending real-time routing messages (before the next channel establishment or channel closing), the real-time delay tables at that moment can be determined from the current real-time traffic load, regardless of intermediate states.

N1's table			N2's table			N3's table			N4's table			N5's table		
D	N	d	D	N	d	D	N	d	D	N	d	D	N	d
2	2	1	1	1	1	1	1	2	2	2	2	3	4	7
2	3	∞	1	4	∞	1	4	∞	2	3	∞	3	3	10
3	3	2	4	4	2	1	5	∞	2	5	∞	4	4	2
3	2	∞	4	1	∞	4	4	5	3	3	5	4	3	15
4	2	3	3	1	3	4	5	12	3	5	12	1	3	12
4	3	7	3	4	7	4	1	∞	3	2	∞	1	4	∞
5	3	12	5	4	4	5	4	7	5	5	2	2	4	4
5	2	∞	5	1	∞	5	5	10	5	3	15	2	3	∞
						5	1	∞	5	2	∞			
						2	1	3	1	2	3			
						2	4	7	1	3	7			
						2	5	∞	1	5	∞			

Table 3: One possible intermediate state of real-time delay tables.

N1's table			N2's table			N3's table			N4's table			N5's table		
D	N	d	D	N	d	D	N	d	D	N	d	D	N	d
2	2	1	1	1	1	1	1	2	2	2	2	3	4	7
2	3	9	1	4	9	1	4	8	2	3	8	3	3	10
3	3	2	4	4	2	1	5	15	2	5	15	4	4	2
3	2	8	4	1	8	4	4	5	3	3	5	4	3	15
4	2	3	3	1	3	4	1	5	3	2	5	1	4	5
4	3	7	3	4	7	4	5	12	3	5	12	1	3	12
5	2	5	5	4	4	5	4	7	5	5	2	2	4	4
5	3	9	5	1	10	5	1	7	5	3	15	2	3	13
						5	5	10	5	2	∞			
						2	1	3	1	2	3			
						2	4	7	1	3	7			
						2	5	14	1	5	14			

Table 4: Steady-state real-time delay tables.

Table 3 represents one possible intermediate state which shows real-time delay tables for all 5 nodes after receiving one real-time routing message from each neighbor. Table 4 shows the steady state of real-time delay tables before the arrival of any real-time channel-establishment request or the termination of any established channel. 2

3.3 The Route-Selection Algorithm

Since the information of all existing channels is necessary for the calculation of a new channel's MWRT as well as for the run-time scheduling of their messages, each node has to maintain another set of tables for the existing channels, one for each outgoing link, in addition to the real-time delay tables. These are called the *tables of existing channels* (TEXCs). Each entry of a TEXC represents a real-time channel which goes through the corresponding link and consists of the following four data fields:

- Channel identifier (ID) which uniquely identifies a real-time channel. To ensure the uniqueness of channel ID, each ID consists of two parts, the source-node address and a channel number (unique within the source node).
- Class ($class$) of the channel.
- Status ($status$) of the channel, established (1) or pending (0).
- The maximum permissible link delay (d) for the channel.

As discussed in Example 1, in order to be consistent with the way channel priorities are assigned for the link-delay estimation [10, 11], these entries are placed in ascending order of d values.

When the source node wishes to establish a real-time channel to another node, say B, it will try to find the current least-MWRT route by considering the traffic of all existing channels. Recall that only those already-established channels are figured in real-time delay tables. Thus, the source node will send a real-time channel-request message (Req) to the next node on the least-MWRT route, which contains a channel ID, the destination address ($dest$), the channel class ($class$), the end-to-end delay bound D, the path ($path$) and the total number of hops ($hops$) this message has traveled thus far, and the accumulated delay d^a. Initially, the d^a field is set to the MWRT of the corresponding outgoing link, $path$ is set to the source node, and $hops$ is set to 1. Although $hops$ is included in the request message for convenience of presentation, it can be omitted in a real implementation because the information carried in $hops$ can be derived from $path$.

Fig. 2 describes the procedure of handling a channel-establishment request after node A receives the request. This procedure can also be applied to the source node by setting $d^a := 0$, $hops := 0$, and $path$ to an empty string. In order to find the next node of a qualified route, Procedure rcv_req uses the destination, B, as an index to the real-time delay table and searches through all routes whose delays to B are not greater than the *remaining* user-required delay bound ($Req.D - Req.d^a$). The **for** loop and the first **if** statement in the loop serve as this function. Since the entries in the real-time delay tables are placed in ascending order of delays to the destination, the search starts from the first entry and can terminate if the current entry cannot satisfy the **if** statement.

After passing the first **if** statement in the loop, if there is no pending channel (the second **if** in the loop), this entry is selected and appropriate actions will be taken by calling Procedure $insert_req$ and $forward_req$. Otherwise, we have to re-compute the MWRT ($= r$) for the corresponding outgoing link, because the real-time delay tables do not account for the pending channels. If the increase of MWRT due to the pending channel

```
Procedure rcv_req
If (Req.dest = A) then {reply_req; return;}
for (i = 1 to number_of_entry(RTDT[Req.dest]) {
        If(Req.d^a + RTDT[Req.dest][i].d) > Req.D) then {send_reply(reject); return;}
        nextnode := RTDT[Req.dest][i].N;
        If (no pending channel in TEXC[nextnode]) then {
          insert_req(TM[nextnode]); forward_req(nextnode); return;
        }
        ;;; compute new MWRT, including established and pending channels.
        r := compute_MWRT(nextnode, 1);
        If (Req.d^a + RTDT[Req.dest][i].d - TM[nextnode] + r ≤ Req.D) then {
          insert_req(r); forward_req(nextnode); return;
        }
}
send_reply(reject);
```

Figure 2: Procedure of processing a channel-establishment request.

($-TM[nextnode]+r$ part in the third **if** statement) doesn't make the delay to the destination greater than $Req.D - Req.d^a$, this entry can be selected as the channel's route. The maximum permissible delay on this link (d field in TEXC) is set to r, instead of obtaining it directly from $TM[nextnode]$. TM is a table of MWRTs (in Section 3.2), and RTDT (in Procedure rcv_req) is used to denote real-time delay tables.

Fig. 3 describes the procedures of inserting a new (pending) channel to TEXC and forwarding a request. As can be seen from these procedures, most fields are directly copied from the establishment-request message to TEXC and the forwarding message.

Fig. 4 shows the operations that a destination will perform after receiving a channel-establishment request. Since the d^a field of a channel-establishment request represents the sum of MWRTs of all links on the path from the source to destination, the user-specified end-to-end delay bound, D, may be larger than d^a, i.e., we are allowed to spend more time than the corresponding MWRT when sending a message over each intermediate link. In such a case, the authors of [10, 11, 15] proposed that $D-d^a$ be divided evenly into *hops* parts by the destination node and distributed to all links along the path. The deadline of a real-time message of this particular channel over an intermediate link is the channel's MWRT of that link plus $(D-d^a)/hops$. Note that one may also choose to divide $D-d^a$ in proportion

Procedure $insert_req(r)$

$TEXC.ID := Req.ID;$
$TEXC(Req.ID).class := Req.class;$
$TEXC(Req.ID).status := pending;$
$TEXC(Req.ID).d := r;$

Procedure $forward_req(nextnode)$

$Req.d^a := Req.d^a + TEXC(Req.ID).d;$
$Req.hops := Req.hops + 1;$
;;; concatenate A and $Req.path$.
$Req.path := A \cdot Req.path;$
;;; all other fields remain the same.
forward this request message to $nextnode$.

Figure 3: Procedures of inserting and forwarding a request.

to each link's MWRT, i.e., the maximum permissible delay over link ℓ is computed as $(D - d^a) \times \frac{MWRT_\ell}{d^a} + MWRT_\ell$, where $MWRT_\ell$ is the MWRT of link ℓ. However, since this method may make the link-delay deadline unnecessarily small over a link which has small MWRT we will adopt the method proposed in [10, 11, 15].

Since this link deadline is stored in the table of existing channels (d field in TEXC) and used for run-time scheduling, $(D - d^a)/hops$ is included in the channel-establishment confirmation message (by procedure $send_reply(accept)$) from the destination to source via the same path the corresponding request message had traveled (but in the opposite direction). Let $Reply$ denote a channel-establishment confirmation message which consists of four fields: ID, $flag$ (accept or reject), $diff$ $(= (D - d^a)/hops)$ and $path$ (the remaining path back to the source node). Fig. 4 describes how a positive confirmation message is constructed ($send_reply(accept)$), and Fig. 5 shows the operations the intermediate nodes will perform when receiving a (positive or negative) reply message ($forward_reply$). Note that $head(list)$ represents the first element of $list$, and $tail(list)$ represents the remaining list after removing $head(list)$ from $list$.

The operations in Procedure $update(node)$ of Fig. 6 are necessary to keep these real-time delay tables and TMs up-to-date after a new channel is established or an existing channel is torn down. Basically, nodes which receive a positive reply to a channel-establishment request will re-compute the MWRT by considering only those established already (including the one just accepted or excluding the one just closed). Based on this new MWRT, real-time routing tables and TM are updated and a new real-time routing message is generated

Procedure *reply_req*

If (*the application accepts the request*) **then** *send_reply*(*accept*);
else *send_reply*(*reject*);

Procedure *send_reply*(*accept*)

$nextnode := head(Req.path)$;
$Reply.ID := Req.ID$;
$Reply.flag := accept$;
$Reply.diff := (Req.D - Req.d^a)/Req.hops$;
$Reply.path := tail(Req.path)$;
send $Reply$ to $nextnode$.

Figure 4: Procedure of processing a channel-establishment request at the destination.

and sent to all neighbor nodes. When a node receives a real-time routing message from a neighbor node, it will follow the procedure described in Section 3.2 to update its real-time delay tables.

The operations necessary to keep real-time delay tables and TMs up-to-date during the channel-disconnect phase are straightforward. We require one of the two communicating peers to send a disconnect message through the route of the real-time channel to the other communicating peer. In the disconnect message, only ID needs to be included. All intermediate nodes and the source node will delete the corresponding entry from their TEXCs and call *update* (Fig. 6) to update both TMs and real-time delay tables. Real-time routing messages may also be sent as discussed in Section 3.2. We will in the next section present examples to illustrate the operations of establishing a channel.

Remark: Since the real-time delay tables go through a period in which the states are indeterminate prior to reaching a steady state, one may wonder how the mechanism for updating delay tables prevents racing conditions from compromising the correctness of the proposed route-selection scheme. The proposed scheme errs on the safe side in handling this problem. In other words, since our scheme requires a confirmation phase, it will never establish a channel that cannot meet the end-to-end delay requirement and hence is always correct. However, our scheme may, albeit for a very short period, reject an establishment request even if the system has the capacity to accept it, due to the temporarily-obsolete information in the real-time routing tables.

Procedure *forward_reply*

If $(Reply.flag = reject)$ **then** delete the entry $TEXC(Reply.ID)$;
else {
$TEXC(Reply.ID).status := established$;
$TEXC(Reply.ID).d := TEXC(Reply.ID).d + Reply.diff$;
insert this entry in the ascending order of d field.
;;; update real-time delay tables if necessary
;;; assume the message is sent/forwarded by node N.
update(N);
}
$nextnode := head(Reply.path)$;
$Reply.path := tail(Reply.path)$;
forward this reply message to the next upstream node $nextnode$.

Figure 5: Procedure of handling reply messages.

3.4 Overhead Analysis

In this section, we analyze the storage overhead of our strategy, since the primary concern of using a table-driven route-selection algorithm is the size of the tables.

The largest table used in the proposed strategy is the real-time delay table. All the other tables are much smaller than the real-time delay table. Thus, we will start with the derivation of an upper bound of the size of the real-time delay table.

Let N denote the number of nodes in the network and a denote the average number of neighbors of a node (or node degree). Then, for each class of real-time channels, the maximum size of all real-time tables in the network is $N \times (N-1) \times a \times 3$. The first N represents the fact that there are N real-time tables per class in the network. $(N-1)a$ represents the number of entries for a real-time delay table and 3 is the number of records for each entry. Basically, the proposed strategy incurs $O(N^2)$ storage overhead for maintaining real-time delay tables.

The table of MWRTs is much smaller than the real-time delay tables. The size of all tables of MWRTs is only $2aN$, where N represents the number of such tables in the network, and there are a entries per table and 2 records per entry.

The size of the above two tables does not change regardless how many real-time channels are established and which routes they use. However, the size of TEXC does depend on the number of existing channels and the average number of hops for each channel. Let h denote

Procedure $update(node)$

$message := \phi$;

for each class $class$ of real-time channels {

;;; compute the new MWRT, including only established channels.

$r := compute_MWRT(node, 0)$;

If $(r \neq TM[node])$ **then** {

$update_RTDT(node, r)$;

$TM[node] := r$;

$table := \cup_{i=all\ destinations}\{(i, RTDT[i][1].d)\}$;

}

$message := message \cup \{(class, table)\}$;

}

send a real-time routing message, $message$, to all neighbor nodes;

Procedure $update_RTDT(node, r)$

for all destinations, $dest$ {

$i := 1$;

while $(i \leq number_of_neighbors)$ {

If $(RTDT[dest][i].N \neq node)$ **then** $i := i + 1$;

else $\{RTDT[dest][i].d := RTDT[dest][i].d + r - TM[node]; break;\}$

}

}

Figure 6: Procedure of updating real-time delay tables.

the average number of hops for an existing channel and n denote the number of existing channels. Then the size of all TEXCs in the network can be specified as $4hn$, where 4 is the number of records per entry (representing an existing channel) in a TEXC. Note that h here is of $O(N)$ since our strategy chooses only loop-free routes.

Based on this analysis, unless the number of existing channels is extremely large (i.e., more than $O(N)$), our strategy incurs an $O(N^2)$ storage overhead for supporting each class of real-time channels.

4 Examples

We will use the network in Fig. 1 as the demonstrative examples in this section.

Example 3: In Example 2, we constructed the real-time delay tables for a class of real-time channels specified by S_{max} = 100 Kbps and p = 33 ms. In this example, we want to establish a channel ($ID = 1:1$) of this class (class 1) with an end-to-end delay bound D = 32 ms from N1 to N5.

Link-bandwidth reservation: N1 will use N5 as the index to its real-time delay table (Table 4), and find the next node, N2, on the least-MWRT path to N5 with MWRT = 5 ms. Since $D > 5$ ms and there is no pending channel, N1 will insert this channel into its TEXC for link $N1 \rightarrow N2$. By looking up the table of MWRTs ($TM[N2] = 1$) for this class, the entry representing this channel in TEXC can be set to $(1:1, 1, 0, 1)$. The channel-establishment request $Req(1:1, N5, 1, 32, N1, 1, 1)$ can also be sent to the next node, N2.

After receiving this request message, N2 will call *rcv_req* to handle this request message and forward it to the next hop on the least-MWRT path to N5. As can be seen from Table 4, N4 is the next hop and the delay is 4 ms. Since the accumulated delay (d^a) carried in the request is 1 ms, i.e., the end-to-end MWRT ($1 + 4 = 5$ ms) is not greater than D = 32 ms and there is no pending channel on link $N2 \rightarrow N4$, this channel request will be inserted into N2's TEXC for link $N2 \rightarrow N4$. Because $TM[N4] = 2$ (for class 1), the entry in TEXC will be set to $(1:1, 1, 0, 2)$. A request message, $Req(1:1, N5, 1, 32, N2N1, 2, 3)$, will also be forwarded to N4.

N4's operations are similar to N2's. The entry inserted in N4's TEXC (link $N4 \rightarrow N5$) for this request is $(1:1, 1, 0, 2)$, and the request message forwarded to N5 is $Req(1:1, N5, 1, 32, N4N2N1, 3, 5)$.

N5's operations will be different from those of N2 and N4, because N5 is the destination of the requested channel. If the peer application at N5 decides to accept this channel request, a positive reply message will be constructed and sent back to N1 via the same path of the request message traveled but in the opposite direction. Since D = 32 ms is greater than the accumulated delay (d^a = 5 ms) carried in the request message, the difference, $D - d^a$, will be divided evenly into *hops* parts and distributed to all links along the path. Thus, using Procedure *send_reply*(*accept*) (Fig. 4), $diff = (32 - 5)/3 = 9$ and the positive reply message will be $Reply(1:1, accept, 9, N2N1)$. This reply message will then be sent to N4 (the head of path carried in the request message).

N1's table		N2's table		N3's table		N4's table		N5's table	
N	MWRT	N	MWRT	N	MWRT	N	MWRT	N	MWRT
2	3	1	3	1	6	2	6	3	30
3	6	4	6	4	15	3	15	4	6
				5	30	5	6		

Table 5: Tables of MWRTs where N stands for Neighbor.

Channel-acceptance confirmation: After receiving the positive reply from N5, N4 will call $forward_reply$ and update the entry in TEXC (link $N4 \rightarrow N5$) representing channel 1:1 to $(1:1,1,1,11)$, where the third component indicates this channel to have been established, and the fourth component $(2+9=11)$ shows the maximum permissible delay of this channel over link $N4 \rightarrow N5$. This positive reply (after removing $head(Reply.path)$) will then be forwarded to the next upstream node (N2) specified by $head(Reply.path)$.

Due to the establishment of channel 1:1, the MWRTs of channels (of all classes) over link $N4 \rightarrow N5$ have to be re-computed. However, by using the method described in Section 3.1, the MWRT for class-1 channels still remains to be 2 ms. Thus, N4's real-time delay tables will remain unchanged and no real-time routing messages will be sent. Note that although the MWRT of class-1 channels does not change, the MWRTs for other classes may change after channel 1:1 is established. Both TMs and real-time delay tables for these classes will be updated and real-time routing messages will also be sent to reflect the new MWRTs of these classes. We will demonstrate the update operations in Example 4.

The operations performed by N2 and N1 are similar to N4's. The TEXC for link $N2 \rightarrow N4$ will contain an entry $(1:1,1,1,11)$ and the TEXC for link $N1 \rightarrow N2$ will contain an entry $(1:1,1,1,10)$. All real-time delay tables and TMs remain unchanged because the establishment of channel 1:1 does not increase the MWRT of class-1 channels over the links of the route.

If the application at the destination node N5 refuses to accept this request, a negative reply will be sent back via the same path. After receiving the negative reply, all nodes will delete the corresponding entry from their TEXCs and forward the reply message to the next upstream node specified by the $path$ in the reply message. 2

Example 4: Another class (class 2) of real-time channels specified by $S_{max} = 300$ Kbps and $p = 20$ ms will be used in this example. Table 5 shows the tables of MWRTs for all 5 nodes after channel 1:1 is established (Example 3), and Table 6 shows the steady state of real-time delay tables for class-2 channels. We will establish a class-2 channel (ID = 1:2, destination = N5, and $D = 30$ ms) and then show the change in both TMs and real-time delay tables after its establishment.

N1's table			N2's table			N3's table			N4's table			N5's table		
D	N	d	D	N	d	D	N	d	D	N	d	D	N	d
2	2	3	1	1	3	1	1	6	2	2	6	3	4	21
2	3	27	1	4	27	1	4	24	2	3	24	3	3	30
3	3	6	4	4	6	1	5	45	2	5	45	4	4	6
3	2	24	4	1	24	4	4	15	3	3	15	4	3	45
4	2	9	3	1	9	4	1	15	3	2	15	1	4	15
4	3	21	3	4	21	4	5	36	3	5	36	1	3	36
5	2	15	5	4	12	5	4	21	5	5	6	2	4	12
5	3	27	5	1	30	5	1	21	5	3	45	2	3	39
						5	5	30	5	2	∞			
						2	1	9	1	2	9			
						2	4	21	1	3	21			
						2	5	42	1	5	42			

Table 6: Steady-state real-time delay tables for class-2 channels.

As in Example 3, the path N1N2N4N5 will be chosen because it is the least-MWRT path from N1 to N5. Thus, a request will be sent from N1 to N5 via this path and each node on this path will insert a corresponding entry into its own TEXC to reflect the existence of channel 1:2. The entries to be inserted into the TEXCs of N1, N2 and N4 are: link $N1 \rightarrow N2$: $(1:2,2,0,3)$, link $N2 \rightarrow N4$: $(1:2,2,0,6)$, and link $N4 \rightarrow N5$: $(1:2,2,0,6)$. If the peer application at the destination node accepts this channel-establishment request, N5 will compute $diff = 30-(3+6+6) = 5$ ms and send a positive $Reply(1:2,1,5,N2N1)$ to N4.

After receiving the positive reply to the request of establishing channel 1:2, N4 will set the status of this channel to "established" and the maximum permissible delay to $6+5 = 11$ ms in the TEXC (link $N4 \rightarrow N5$). Then, N4 will re-compute the MWRT of both classes over link $N4 \rightarrow N5$, including only the load of established channels. For class-2 channels, $TM[N5]$ is then modified to 14, because any new channel request of class 2 has to be given priority lower than both channel 1:1 and 1:2 in order to retain the performance guarantees of existing channels. However, the MWRT of class-1 channels does not change. Thus, the real-time delay tables will be updated (Table 7) and real-time routing messages containing only the class-2 information will be sent to N4's neighbors. For example, the message sent to N2 will contain three entries: (N5,14), (N3,15), (N1,21), and the message sent to N3 will also contain three entries: (N5,14), (N2,6), and (N1,9). Note that the third entry in the message to N2 is the result of loop-free routing. Since the least-MWRT path from N4 to N1 is N4N2N1 which includes N2, the second best path N4N3N1 is sent to N2 as the (loop-free) least-MWRT path.

The positive reply will then be forwarded to N2 then to N1. The operations of N2 are similar to those of N4. The MWRTs of both classes over link $N2 \rightarrow N4$ have to be re-computed and the real-time delay tables have to be updated accordingly. The MWRT of

N1's table			N2's table			N3's table			N4's table			N5's table		
D	N	d	D	N	d	D	N	d	D	N	d	D	N	d
2	2	3	1	1	3	1	1	6	2	2	6	3	4	21
2	3	27	1	4	35	1	4	24	2	3	24	3	3	30
3	3	6	4	4	14	1	5	45	2	5	53	4	4	6
3	2	32	4	1	24	4	4	15	3	3	15	4	3	45
4	2	17	3	1	9	4	1	23	3	2	15	1	4	15
4	3	21	3	4	29	4	5	36	3	5	44	1	3	36
5	2	31	5	4	28	5	4	29	5	5	14	2	4	12
5	3	35	5	1	38	5	5	30	5	3	45	2	3	39
						5	1	37	5	2	∞			
						2	1	9	1	2	9			
						2	4	21	1	3	21			
						2	5	42	1	5	50			

Table 7: Steady-state real-time delay tables for class-2 channels.

class 2 over link $N2 \rightarrow N4$ will increase to 14 and that of class 1 remains unchanged. Thus, {(N5,28),(N4,14),(N3,29)} will be sent to N1, and {(N1,3),(N3,9),(N5,38)} will be sent to N4 as the real-time routing messages for class-2 channels.

After receiving the confirmation of establishing channel 1:2, N1 will also re-compute the MWRTs of both classes. Since both MWRTs do not change over link $N1 \rightarrow N2$, no further actions are necessary.

In addition to the positive reply messages, the real-time routing messages generated by N4 and N2 will also be used to update real-time routing tables by those nodes which receive them as discussed in Section 3.2. Following the procedure of building real-time delay tables, Table 7 shows the steady-state real-time delay tables for all 5 nodes after channel 1:2 is established.

5 Conclusion

In this paper, we have proposed a table-driven distributed route-selection scheme which is guaranteed to find a qualified route, if any, for each real-time channel-establishment request. By equipping a real-time delay table with each node, the proposed scheme can choose a route for each real-time channel requested by a table look-up.

The proposed scheme is presented in procedure form, and is also expected to make significant performance improvements over centralized and other distributed schemes because the route-selection problem can be solved by just a simple table look-up. Several examples are presented to illustrate how the proposed scheme works.

References

[1] D. P. Anderson, S. Y. Tzou, R. Wahbe, R. Govindan, and M. Andrews, "Support for continuous media in the dash system," *Proc. 10-th Int'l. Conf. on Distributed Computing Systems*, pp. 54–61, May 1990.

[2] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall International, 1987.

[3] C. C. Cheng, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves, "A distributed loop-free routing algorithm suitable for arbitrary link weights," Technical Report CSS-89-05, Dept. Electrical Engineering and Computer Science, Northwestern Univ., September 1989.

[4] C.-C. Chou and K. G. Shin. *Multiplexing Statistical Real-Time Channels on a Multiaccess Network*. Submitted for publication.

[5] C.-C. Chou and K. G. Shin. *Statistical Real-Time Channels on Multiaccess Networks*. Submitted for publication.

[6] C.-C. Chou and K. G. Shin, "Statistical real-time video channels over a multiaccess network," *Proc. High-Speed Networking and Multimedia Computing Symposium,IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, February 1994.

[7] 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.

[8] 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, April 1990.

[9] D. D. Kandlur and K. G. Shin, "Design of a communication subsystem for HARTS," Technical Report CSE-TR-109-91, CSE Division, Department of Electrical Engineering and Computer Science, The University of Michigan, 1991.

[10] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multi-hop networks," *Proc. 11-th Int'l. Conf. on Distributed Computing Systems*, pp. 300–307, 1991.

[11] D. D. Kandlur, *Networking in Distributed Real-Time Systems*, PhD thesis, University of Michigan, 1991.

[12] P. Pancha and M. E. Zarki, "A look at the MPEG video coding standard for variable bit rate video transmission," *INFOCOM*, 1992.

[13] K. G. Shin and M.-S. Chen, "Performance analysis of distributed routing strategies free of ping-pong-type looping," *IEEE Trans. Comput.*, vol. C-36, no. 7, pp. 129–137, February 1987.

[14] K. G. Shin and M.-S. Chen, "Minimal order loop-free routing strategy," *IEEE Trans. Comput.*, vol. C-39, no. 5, pp. 870–888, July 1990.

[15] K. G. Shin and C.-C. Chou. *A Distributed Route-Selection Scheme for Establishing Real-Time Channels*. Submitted for publication.

[16] K. G. Shin and C.-C. Chou, "Design and evaluation of real-time communication for FieldBus based manufacturing systems," *1992 IEEE Local Computer Network Symposium*, pp. 483–492, September 1992.

[17] K. G. Shin and C.-C. Chou, "A simple distributed loop-free routing strategy for computer communication networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1308–1319, December 1993.

[18] D. C. Verma, *Guaranteed Performance Communication in High Speed Networks*, PhD thesis, University of California at Berkeley, 1991.

[19] J. Walrand, *Communication Networks: A First Course*, Irwin and Aksen Associates, 1991.