Distributed Route Selection for Establishing Real-Time Channels

Kang G. Shin, Chih-Che Chou, and Seok-Kyu Kweon

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
Email: {kgshin,ccchou,skkweon}@eecs.umich.edu
Phones: 734-763-0391 (voice); 734-763-4617 (Fax)

August 30, 1999

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 message belonging to the connection. 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 an efficient distributed route-selection scheme that is guaranteed to find a "qualified" route, if any, satisfying the performance requirement of the requested channel without compromising any of the existing guarantees. The proposed scheme can also eliminate the common reliability/performance bottleneck of a centralized route-selection scheme, while improving efficiency over the centralized and other distributed schemes. Although the proposed solution starts with searching all possible routes *in parallel*, it prunes infeasible routes quickly and its worst-case operational overhead is shown to be only a linear function of the number of links in the network. Several examples and simulation results are presented to demonstrate the effectiveness of the proposed distributed route-selection scheme as compared to sequential route-search schemes.

Key Words — Hard real-time communication systems, real-time channel/connection, deadline guarantees, distributed route selection, message scheduling.

The work described in this paper was supported in part by the Office of Naval Research under Grants N00014-J-94-1-0229 and N00014-99-1-0465. 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 agency.

1 Introduction

The demand of real-time network services has been rising fast in many applications, such as process control, factory automation, and nuclear reactor control. To meet this demand, researchers proposed several real-time communication protocols [9, 11, 19]. The concept of "real-time channel" proposed by Ferrari and Verma [9] and refined by Kandlur *et al.* [11] is one of the most notable solutions to the problem of meeting message/packet[1] delivery deadlines in wide area networks (WANs). 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 [9].

Generally, two distinct phases are required to realize the concept of real-time channel: off-line channel establishment and run-time message scheduling. The channel-establishment phase is of prime importance to the realization of a real-time channel, and during this 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 [3–5, 9–11, 15], very few of them have addressed explicitly the issue of selecting a route between the source and destination of a channel with end-to-end deadline guarantees, despite its importance to the channel-establishment phase.

Since the number of possible routes between two communicating peers 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 select 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. 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 based on the centralized approach [4, 5, 10, 11, 15]. 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 network thus far, and hence, can select an appropriate route for each requested real-time channel. In such a centralized scheme, all of the real-time channel-establishment requests require the network manager's approval. That is, each channel-establishment request is sent, along with

[1] We will use the term "message" throughout the paper, but it could be replaced by the term "packet", depending on the desired context.

its traffic-generation characteristics and user-specified performance requirements, 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 with the centralized approach one can devise efficient algorithms for the network manager to select qualified routes, there are two serious problems with this approach. 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, no new real-time channel can be established.

To eliminate/alleviate performance and reliability bottlenecks, one may use distributed link-state routing [6]. In this approach, each node maintains the information on the entire network at a local database by exchanging link-state information with all other nodes, so that the source or the destination can determine a route for a new requested channel solely based on the information kept in its database. The original link-state routing [6] was intended for best-effort communication in a datagram network, and incurs a very small overhead because each node is required to exchange minimal information on the status of neighboring links periodically or occasionally. In order for this type of routing protocols to be used for real-time communication, however, the detailed information on real-time channels traversing each link must be exchanged between nodes; this is significantly more than the minimal information exchanged in the original protocol, thus consuming significant network resources [9, 11]. For example, for the real-time channel approach in [11] — which is based on the linear-bounded arrival process — local delay bounds, minimum message inter-arrival times and maximum message sizes of all the real-time connections established over each link must be broadcast over the entire network whenever a new channel is established or an old channel is torn down. In this type of real-time communication protocols, the aggregation of link-state information [14] itself is impossible. This excessive information exchange and the large database kept at each node make it impractical to use distributed link-state routing.

In this paper, we propose an efficient distributed route-selection scheme which is guaranteed to find a qualified route, if any, for each real-time channel-establishment request. The proposed scheme searches for a qualified route in parallel by flooding the connection request messages through the network. In order to reduce the number of request messages, it prunes infeasible routes quickly. Unlike the centralized approach, our scheme doesn't suffer performance and reliability bottlenecks. Moreover, it doesn't require each node to keep the global information on the network topology, and can also keep the overhead in establishing a channel very low even under a heavy-load condition. This scheme also works well for handling multiple *simultaneous* channel-establishment requests.

The paper is organized as follows. Section 2 states the problem of finding a qualified route for each channel-establishment request. Our proposed solution to this problem and its overhead analysis are presented in Section 3. In Sections 4 and 5 and in the Appendix, we demonstrate, using examples and simulations, the correctness and the effectiveness of the proposed solution. The paper concludes with Section 6.

2 Distributed Route-Selection Problem

The reliability and performance bottlenecks of the centralized route-selection scheme make the distributed approach an attractive alternative. To choose a qualified route for a new real-time channel, we have to perform an admission test on each candidate route to check 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 meeting the user-specified end-to-end delay requirement.)

Since there could be a large number of routes between two communicating peers, choosing a qualified route among all possible routes between the source and destination of each requested channel may not be an easy task. One can think of two simple-minded approaches to the distributed route-selection problem:

1. Sequential search of n possible routes one by one until a qualified one is found or all the n routes are exhausted.
2. Parallel search of all possible routes, i.e., sending multiple copies of an establishment-request message through all possible routes, making "conditional" resource reservation and performing admission tests on all of them.

The first approach determines a candidate route using the information on the network topology and sends a real-time channel-establishment request along the candidate route. If channel-admission tests succeed at all the links along the route, the channel will be established. Otherwise, a second candidate is chosen with the first one excluded, and the same procedure is repeated. Since choosing candidate routes requires the knowledge of the network topology, this approach can be considered as a distributed approach augmented with a centralized feature. The well-known Dijkstra shortest-path algorithm [2] can be used for finding candidate routes. However, this approach could be potentially very time-consuming for the complete search of all possible routes, and its operational overhead which is measured in number of connection establishment request messages is proportional to n. One way to reduce the search cost is to (i) determine several minimum-hop routes (MHRs) and (ii) check them, one at a time, if they can meet the connection's timing requirements, until a "qualified" route is found from the selected MHRs or all MHRs are exhausted.

The second approach, on the other hand, may be very fast although it may possibly induce a large operational overhead. Moreover, it can guarantee the discovery of a qualified route, if any. This approach, if its overhead can be reduced/minimized, will therefore become very attractive.

We propose a scheme based on the second approach that is guaranteed to find a qualified route, if any, for the case of a single establishment request at a time, while minimizing its operational overhead. The scheme is also designed to work well for multiple simultaneous requests by employing 'temporary' resource reservation. In this scheme, each node in the network maintains certain information of the real-time traffic going through it and exchanges this information with its neighbors, so that an algorithm similar to the Bellman-Ford shortest path algorithm [2, 17] can be used to find a qualified route. Although the proposed scheme starts with searching all possible routes *in parallel*, it prunes infeasible and/or inferior routes very quickly, thereby reducing the operational overhead, i.e., the number of request messages. Under the realistic assumption that messages travel faster through lightly-loaded links, its operational overhead is only a linear function of E, the number of links in the network. Another feature of the proposed approach is elimination of shortest-path calculations from the routing algorithm, thanks to its parallel search.

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, 11, 12, 16], the generation of real-time messages is assumed to be governed by the 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, the number of messages generated in any time interval of length t does not exceed $B_{max} + tR_{max}$. Based on this message arrival model, the authors of [11, 12] proposed a scheme to estimate the worst-case delay on each link and a run-time scheduling algorithm for real-time messages. 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 request. Note that these schemes have been developed under the assumption that a proper route for the requested channel was already available. Using the delay-estimation method in [11, 12] and a Bellman-Ford-like algorithm, we will in this paper develop a scheme to find a qualified route for each channel-establishment request. Table 1 lists the notations used in this section.

S_{max}	maximum message size
R_{max}	maximum message rate
B_{max}	maximum burst size
M_i	i^{th} channel
C_i	maximum time required to transmit a message of channel M_i on a link
p_i	minimum message inter-arrival time in M_i
d_i	maximum delay assigned to M_i on a link
C	maximum service time for a message at a link
p	minimum message inter-arrival time
d	maximum permissible link delay
d^a	accumulated delay from the source to the current node
D	user-specified end-to-end delay bound
$timeout$	expiration time of a request message
r	MWRT of a link
$path$	a route which a request message has traveled thus far
$hops$	the total number of hops a request message has traveled thus far
$flag$	indicator of acceptance or rejection
$diff$	$(D - d^a)/hops$

Table 1: Notations

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 the underlying real-time traffic. Since the algorithm in [11, 12] will be used to estimate link delays, we will briefly introduce this algorithm first.

The goal of the algorithm in [11, 12] is to compute the minimum worst-case delay on a link 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, $p_i = \frac{1}{R^i_{max}}$ is the minimum message inter-arrival time in M_i, and d_i is the maximum delay assigned to M_i on this link, or *link (delay) deadline*. Note that the inequality $d_i \leq p_i$ must hold for the algorithm in [11, 12] to work correctly. Given a new channel $M_{k+1} = (C_{k+1}, p_{k+1})$ to be established, the authors of [11, 12] proposed an algorithm for computing the minimum worst-case response time (MWRT), r_{k+1}, on a link of channel M_{k+1}'s route without compromising the performance guarantees of other existing channels over the link. The algorithm statically assigns priority to each real-time channel to calculate the MWRT for this new channel, but uses a multi-class Earliest-Due-Date (EDD) algorithm for run-time scheduling. The algorithm can compute the MWRT for a new channel through 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 [11, 12] did *not* include those channels pending for final confirmation in the calculation of MWRT for the new channel-establishment request, but we will include them in our calculation of MWRT as if they had already been established. This can simplify the channel-establishment phase, since the MWRT remains 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 MWRT larger than what it actually would be if some of them are rejected or choose not to use this link later. This over-estimation of MWRT may sometimes result in incorrect rejections of channel-establishment requests. The over-estimation problem occurs only when multiple requests are initiated at about the same time. The incorrect rejection decisions due to the over-estimation of MWRT will be made only when there is a very high percentage of real-time traffic so that the over-estimation of MWRT 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 henceforth be referred to as *link-delay estimation*.

3.2 The Route-Selection Algorithm

Based on the above definition of link delay, we can apply a variation of the Bellman-Ford algorithm [2, 17] to solve the route-selection problem. The proposed algorithm is not exactly the same as the original Bellman-Ford shortest path algorithm in terms of the number of routes explored. Under the original Bellman-Ford algorithm, only the one which has the shortest delay is explored at any time. However, under our algorithm, we simultaneously explore *all* routes which could possibly be shortest.

Since the information of existing channels is necessary for the calculation of a new channel's

MWRT as well as for the run-time scheduling of messages belonging to those channels already established, each node has to maintain two sets of tables for existing channels. The first set is the tables of established channels (TECs), one for each of its outgoing links. Each entry of a TEC 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 the corresponding real-time channel. In order for a source node to generate unique channel IDs, each ID consists of two parts. The first part is the source ID (or address), and the second part is a channel number (unique within the source). This composition of channel IDs ensures their uniqueness throughout the network.
- The maximum service time of a message (C) of this channel.
- The minimum message inter-arrival time (p) of this channel.
- The maximum permissible delay on this link (d) for this channel.

To be consistent with the way channel priorities are assigned for the link-delay estimation, these entries are placed in ascending order of d values, i.e., the highest priority is given to the channel with the least permissible delay on this link. Note that this priority assignment is used only for calculating 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.)

The second set of tables each node has to maintain are "temporary" tables for pending channel-establishment requests, also one for each of its outgoing links. These tables will be referred to as "tables of pending requests" (TPRs). Each entry of a TPR represents a channel-establishment request (or a pending channel) and consists of six fields. The first three fields are the same as those of a TEC and the remaining three fields are:

- d^a: the accumulated delay from the source to the current node,
- *timeout*: the expiration time of this request message,
- r: MWRT of the corresponding outgoing link.

For a real-time channel-establishment request, the first five fields are the same for all outgoing links of a node, and thus, can be shared among all TPRs for the node's different outgoing links, i.e., one may easily use only one table to store the combined information of all TPRs. However, for convenience of presentation, we will assume that each TPR (one per outgoing link) contains all of these data fields.

When the source wishes to establish a real-time channel to another node, it will use the foregoing link-delay estimation method to compute the channel's MWRT on each of its outgoing links. After computing all MWRTs, the source will send a real-time channel request message (Req) via each outgoing link, which contains a channel identifier (ID), the destination address (*destination*), the maximum message size of this channel (S_{max}), the minimum message inter-arrival time (p), the end-to-end delay bound D, the expiration time (*timeout*) of this request, the path (*path*) and the total number of hops (*hops*) this message has traveled thus far, and the corresponding accumulated delay d^a. Initially, the d^a field is set to the MWRT of the corresponding link, *path* is set to the source and *hops* is set to 1. Note that although we include *hops* 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*. Copies of this channel-request message will be put into the queues of all of its outgoing links at the same time,[2] each with priority lower than all existing channels but higher than non-real-time traffic. This new establishment request will also be inserted into the source node's TPRs.

Fig. 1 outlines the procedure an intermediate node A will execute upon receipt of a real-time channel-establishment request. Procedure *rcv_req* checks the received request message to determine

Procedure rcv_req

If $(Req.timeout \leq current\_time)$ **then** $discard\_req$;

else if $(Req.ID \in TEC)$ **then** $discard\_req$;

else if $(Req.destination = A)$ **then** $reply\_req$;

else if $(Req.ID \in TPR)$ **then** {
    if $(Req.d^a \geq TPR(Req.ID).d^a)$ **then** $discard\_req$;
    else {
        $TPR(Req.ID).d^a := Req.d^a$; $forward\_req$;
    }
}

else {   // $Req.ID$ *not in* TPR
    $r := compute\_MWRT$;
    if $(Req.d^a + r < Req.D)$ **then** { $insert\_req(r)$; $forward\_req$;}
    else $discard\_req$;
}

Figure 1: Procedure for processing a channel-establishment request

whether the message should be discarded or processed further. The first two **if** statements check whether the request has expired ($timeout \leq current\_time$), or the request is in any of TECs, i.e., a qualified route for the channel has already been found. If either of these is true, the request message will be discarded. Procedure *reply_req* will then be called in if node A is the destination of the

[2] One can build hardware to do this [8]. If such hardware is not available then the copies will be put in the queues sequentially, one at a time.

channel.

The fourth **if** clause is for the requests already in TPRs. The request will be discarded if its accumulated delay (d^a) is not smaller than the corresponding one in TPRs which represents the minimum accumulated delay from the source known to node A thus far, i.e., the path this request message has traveled so far is *inferior* to the ones recorded in TPRs. If the d^a value of the received request is smaller, node A will update its TPRs to reflect the fact that a better route has been found and the received request will be forwarded to the next node by procedure $forward\_req$.

Finally, we conclude that the request is new to node A. Thus, the request message will be appropriately stored and forwarded (with procedure $insert\_req$ and $forward\_req$) if the sum of the accumulated delay and the MWRT of the corresponding next link is less than D. Otherwise, the path this request message has traveled so far cannot possibly be a qualified one — i.e., the path is *infeasible*, — so it will be discarded. A request message will be forwarded by an intermediate node only if it carries a smaller accumulated worst-case response time ($Req.d^a$) before its expiration time.

Fig. 2 shows the procedures of inserting a new channel-establishment request and forwarding a request. As can be seen from these procedures, most of the fields are directly copied from the requesting messages to TPR and the forwarding messages. Note that the request message is assumed to come from the immediate upstream node B.

Procedure $insert\_req(r)$

$TPR.ID := Req.ID;$
$TPR(Req.ID).C := Req.S_{max}/link\_speed;$
$TPR(Req.ID).p := Req.p;$
$TPR(Req.ID).d^a := Req.d^a;$
$TPR(Req.ID).timeout = Req.timeout;$
$TPR(Req.ID).r := r;$

Procedure $forward\_req$

$Req.d^a := TPR(Req.ID).d^a + TPR(Req.ID).r;$
$Req.hops := Req.hops + 1;$

concatenate A and $Req.path$.
$Req.path := A \cdot Req.path;$

all other fields remain the same.
forward this request message to all neighbors except B.

Figure 2: Procedures for inserting and forwarding a request

Each destination node has to keep a temporary list of already-processed requests (LPRs) in order to avoid reporting the request to applications more than once. Each entry of this list consists of two fields, request ID and *timeout* which tells when to discard the request. Fig. 3 shows the operations a destination node will perform after receiving a request. From Procedure *reply_req*, one can see that if the system decides to accept the request, the (qualified) path carried by the request arrived first will be selected as the route for the real-time channel.

Procedure *reply_req*

If ($Req.ID \in LPR$) **then** *discard_req*;

else {
insert a new entry to LPR
$LPR.ID := Req.ID$;
$LPR(Req.ID).timeout := Req.timeout$;
If (*the application accepts the request*) **then** *send_reply*(*accept*);
else *send_reply*(*reject*);

}

Figure 3: Procedure for processing a channel-establishment request at the destination

Since the d^a field of a 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 MWRTs when sending a message across each intermediate link. In such a case $D - d^a$ is divided evenly for all the intermediate links of the real-time channel's path [11, 12]. Then, the permissible delay of a real-time message of this particular channel over an intermediate link — simply called the *link (delay) deadline* — is the channel's MWRT of that link plus $(D - d^a)/hops$. Since this sum is stored in the table of existing channels (d field in TEC) 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 the source via the same path the corresponding request message had traveled (but in the opposite direction). Let *Reply* denote a 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 shows how a positive confirmation message is constructed (*send_reply*(*accept*)), and 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 *head*(*list*) is removed from *list*.

The operations necessary to keep these route selection tables (mainly TECs) up-to-date during the channel-disconnect phase are very simple. 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 this disconnect message, only the channel ID needs to be included. All the intermediate nodes will delete the corresponding entries in their TECs upon receiving the disconnection message. Thus, we do not consider the load of this real-time channel in all subsequent MWRT estimations.

Procedure $send\_reply(accept)$

$Reply.ID := Req.ID;$

$Reply.diff := (Req.D - Req.d^a)/Req.hops;$

$Reply.flag := 1;$

$Reply.path := tail(Req.path);$

Procedure $forward\_reply$

If $(Reply.flag = 0)$ **then** $TPR(Reply.ID).d^a = 0;$

else {

move the channel from TPR to TEC

copy the ID, C and p fields from TPR to TEC.

$TEC(Reply.ID).d := Reply.diff + TPR(Reply.ID).r;$

delete the corresponding entries in all TPRs of node A.

}

$next := head(Reply.path);$

$Reply.path := tail(Reply.path);$

forward this reply message to the next upstream node *next*.

Figure 4: Procedure for handling reply messages

3.3 Performance and Overhead Analysis

The first goodness measure we are interested in is the "completeness" of the proposed scheme, i.e., whether the scheme is capable of finding a qualified route, if any. For a single request, the Bellman-Ford (shortest-delay path) [2, 17] algorithm ensures that the least MWRT path will be found, although other larger-delay routes may be found first when the number of hops of these routes are smaller than that of the least MWRT path. If the least MWRT path is not "qualified" for the requested channel, then there is no qualified route available for the channel. Since the least MWRT path can always be found with the Bellman-Ford algorithm, the proposed scheme is complete for the single-request case. However, due to the over-estimation of MWRTs when there are multiple simultaneous requests, the proposed scheme may not be complete, especially when the network has a high percentage of real-time traffic so that the over-estimation of MWRTs will make the end-to-end MWRT delay larger than the user-specified end-to-end delay bound. (As we discussed in the delay-estimation procedure, however, the over-estimation problem is usually not serious in practice. In the Appendix, we will provide an example to illustrate an over-estimation situation.)

Another performance measure is the time to establish a channel. For a single request, the worst-case time needed to accept an establishment request is the time for the request message to travel from the source to the destination then back to the source node via the least MWRT path. This is a round trip delay between the source and destination via the least MWRT path. In general, if a qualified path can be found (may not necessarily be the least MWRT path), the time to complete the corresponding channel-establishment request is the time for the request message to travel from the source to the destination then back to the source via the qualified route found first.

The primary overhead incurred in the proposed channel-establishment procedure is the number of times (copies) a request message has to be transmitted for each channel establishment request. Note that "one time (copy)" is defined as "sending a message across one link", e.g., a message is said to be transmitted n times if the message is sent n hops. For more accurate estimation, the request message is assumed to travel faster through a lightly-loaded link (considering only real-time traffic). This assumption generally holds as the priority of the request message is lower than real-time traffic but higher than non-real-time traffic. Under this assumption, each node is likely to send a request message to its neighbors only once, since the request message will be forwarded only when the conditional statement ($Req.d^a \geq TPR(Req.ID).d^a$) in Fig. 1 is false. A node will likely receive the copy of a request message which travels through the route with the smallest value of $Req.d^a$ first, and thus, it will discard all subsequently-arriving copies of the same request message. A request message will therefore be transmitted at most $2K$ times in this case, i.e., each node sends a copy of the request message to all its neighbors once, where K is the number of links in the network.

In some cases, however, depending on network traffic condition, multiple requests can arrive at a node from different routes. This happens when a request message with a larger $Req.d^a$ arrives earlier at a node along a route which was momentarily lightly-loaded even though large number of real-time channels are already established over some links of the route. In that case, the number of messages that must be received in the entire network could be greater than $2K$. Although it is a rare case, the overhead can be quite large in the worst case. We may be able to prevent the overhead from increasing absurdly in the worst-case by imposing an upper limit to the number of request messages for a real-time channel that can be transmitted through a link or by controlling the *Req.timeout* field of a request message.

A more fundamental way to reduce the overhead in our approach is the pruning function. That is, the conditional statement ($Req.d^a + r < Req.D$) in Fig. 1 is used to terminate the unnecessary propagation of a request to a region where no qualified routes exist. Since a request will be inserted in TPR and forwarded through an outgoing link only when the sum of $Req.d^a$ and the MWRT over the outgoing link is smaller than D ($=Req.D$), the request message will not propagate too far, i.e., the nodes whose distance from the source (in terms of the worst-case delivery delay for this new channel) is greater than D will not receive the request message. Although this fact may not improve the *worst-case* overhead, for most of the time it makes a significant reduction of overhead.

We may also put a restriction on *hops* to reduce the overhead; stop forwarding a request if the number of hops the message has traveled so far is more than a pre-defined limit. Nodes which are located too far (in terms of hop count) from the source N will not receive the requests sent by N. If the pre-defined limit can be chosen appropriately, this method may also significantly improve both the worst-case overhead and the actual performance. However, since this method will also reduce the chance of finding a qualified route, it is not included in the proposed solution and only mentioned as a possible way to reduce the overhead.

Although we argued that our approach has a potential to accompany large overhead and presented ways to reduce it, we still believe that the size of the overhead is justifiable due to the following reasons. First, the number of request messages which are transmitted for establishing a real-time channel is very small compared to the number of messages which are transmitted through the established real-time since, in general, the lifetime of the real-time is much longer than the channel establishment time. Secondly, although the overhead is quite large in our approach, it is much smaller than that of the distributed version of the centralized approach mentioned in Section 1. In the distributed version based on the Link-State Routing, the exchange information must include all the details of all the real-time channels at every node in the entire network, i.e., local delay bound, the minimum message inter-arrival time and the maximum message size, and this information must be broadcast to all the nodes whenever a new channel is established and an old channel is disconnected. In addition, each node must manage a huge database in order to keep track of all this information. Moreover, the consistency problem must be dealt with in order to enable all the nodes to share the identical data. On the other hand, in our approach, each node has only to keep the information on the real-time channels going through the node, and the information contained in the request message is only on the new real-time channel concerned and hence, much smaller than that required in the distributed version. In terms of reliability, our approach is more robust than the distributed version since a failed node or a lost link-status message will worsen the consistency problem in the distributed version unlike ours.

Although the sequential approach mentioned in Section 2 has possibly a smaller overhead, it is less robust than our approach in terms of reliability and has the consistency problem because every node is required to maintain the network topology information.

4 Example

In this section and the Appendix, we will present several examples to illustrate the operations and utility of the proposed route-selection scheme under various conditions.

Example 1: Fig. 5 shows a small network with five nodes. Each link is labeled with a number representing its transmission bandwidth in million bits per second (Mbps); for example, the link between N1 and N2 is a 100 Mbps full-duplex link. We will illustrate all the operations in the network for establishing the first channel with $S_{max} = 50Kb$ (maximum message size), $p = 50\, ms$ (minimum inter-arrival time), and $D = 100\, ms$ (end-to-end delay bound). The source of this channel is N1 and its destination is N5. We thus assign 1:1 as the channel's ID. The first "1" represents the source node, and the second "1" represents a number which is unique to the source node N1. Since

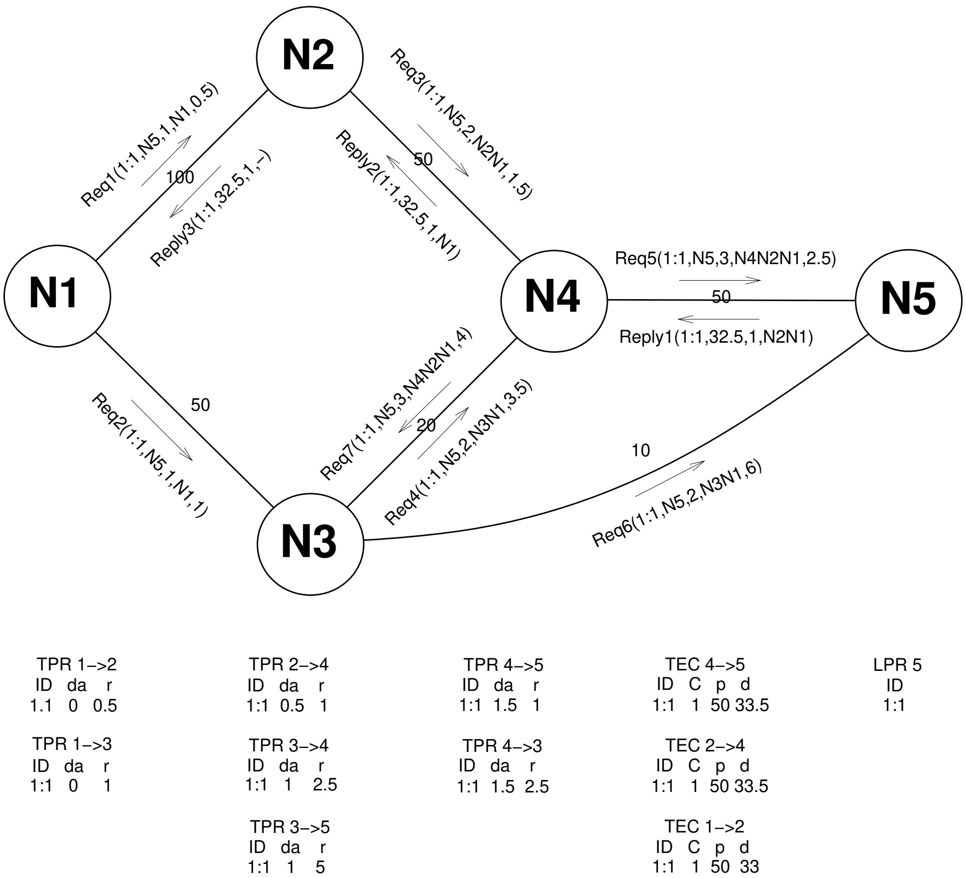


Figure 5: Example 1: a simple five-node network.

no real-time channel exists at this time, all TECs, TPRs, and LPRs are empty.

Upon reception of a channel-establishment request, the maximum service time (C) for the messages of this channel can be computed by dividing the maximum message size by the link bandwidth. $C = 0.5\,ms$ for link $1 \rightarrow 2$ and $C = 1\,ms$ for link $1 \rightarrow 3$. Since there is no other real-time channel, the MWRT on a link equals its C value. N1 stores this request in its TPR:$1 \rightarrow 2$ and TPR:$1 \rightarrow 3$. In Fig. 5, we omit C and p fields because $C = r$ and p are the same in all TPRs. *Timeout* is also omitted for clarity of presentation.

N1 sends two request messages (Req1 and Req2) to N2 and N3, respectively. Although there are 9 fields in a request message, only ID, *destination*, *hops*, *path* and $d^a(ms)$ (in this order) are shown in Fig. 5, since other fields are the same for all request messages. For example, Req1(1:1,N5,1,N1,0.5) represents ID =1:1, *destination* $= N5$, *hops* $= 1$, *path* $= N1$ and $d^a = 0.5\,ms$.

After N2 and N3 receive Req1 and Req2, respectively, these messages will be stored and the C field for link $2 \rightarrow 4$, link $3 \rightarrow 4$ and link $3 \rightarrow 5$ will be computed. Since there is no other channel, each C is equal to the MWRT on the corresponding link. Thus, the entries for this message in TPR:$2 \rightarrow 4$, TPR:$3 \rightarrow 4$ and TPR:$3 \rightarrow 5$ can be inserted and the corresponding request messages can also be sent. As can be seen in Fig. 5, *path* "grows" to N2N1, *hops* increments by 1 and d^a becomes 1.5 in Req3, which is sent to N4 by N2. Req4 (N3 $\rightarrow$ N4) and Req6 (N3 $\rightarrow$ N5) can also be generated in the same manner.

Due to the randomness of network traffic, it is not certain whether Req3 or Req4 will arrive at N4 first. However, since the path N1N2N4 has a smaller MWRT and channel-establishment requests are given priority over non real-time traffic, Req3 will very likely arrive at N4 first. So, in Fig. 5, we assume Req3 arrives at N4 before Req4. N4 will thus process the request based on Req3 and the entries in TPR:4 $\rightarrow$ 5 and TPR:4 $\rightarrow$ 3 can be obtained (as shown in Fig. 5) and the corresponding request messages (Req5 and Req7) can be sent.

When Req4 arrives at N4, the d^a carried in Req4 will be compared with the d^a value stored in N4's TPRs. Since the d^a (=3.5) carried by Req4 is greater than that (=1.5) of N4's TPRs, Req4 will be discarded.

At N5, a similar situation will occur. We assume that Req5 will arrive at N5 before Req6, because Req5 travels through a path of a smaller MWRT. (We would like to stress, however, Req6 may possibly arrive at N5 before Req5.) Since N5 is the destination of the requested channel and the ID carried by Req5 is not in N5's LPR, the channel's ID will be inserted into N5's LPR and a confirmation/reject message is sent back via the path carried in the request message (N4N2N1). If the reply is a confirmation, we need to compute $diff = (100 - 2.5)/3 = 32.5$. In Fig. 5, Reply1 shows a confirmation sent to N4 by N5 with ID =1:1, $flag = 1$, $diff = 32.5$ and $path$ =N2N1.

After receiving Reply1, N4 inserts an entry with ID =1:1, $C = 1\,ms$, $p = 50\,ms$, and $d = 33.5$ (=32.5+1) into TEC:4 $\rightarrow$ 5. Then all the corresponding entries (with ID =1:1) in N4's TPRs are deleted and Reply2 (with $path = tail(path)$) is sent to the next node specified in Reply1's $head(path)$. Operations in N2 and N1 are similar. $\square$

See the Appendix for more complex examples.

5 Simulation and Discussion

In order to comparatively evaluate the proposed scheme, we have conducted an in-depth simulation study. In this study, we measured and compared the probability of successfully establishing channels under various load conditions using (i) the proposed (parallel-search) approach and (ii) sequential-search approaches based on the Shortest-Path-First (SPF) algorithm. All the hops are assumed to have the same weight/cost, and thus, SPF practically finds the route with the minimum number of hops. We also evaluated their operational overhead and time to establish a real-time channel. We first describe the simulation model then the simulation results.

5.1 The Simulation Model

We used a part of the MBONE network in the North America region in Fig. 6 as the network topology in order to evaluate the performance of the proposed scheme in the WAN environment. All T3 links and those nodes connected via T3 links are included in the simulation. We also used two additional network topologies in our simulation in order to emphasize the usefulness of our approach in networks with better connectivity (e.g., backbone networks). Fig. 7 shows these two additional network topologies used in our simulation. Fig. 7(a) is an ordinary rectangular (unwrapped) mesh and Fig. 7(b) is a wrapped rectangular mesh, both consisting of 25 nodes each. We assume that

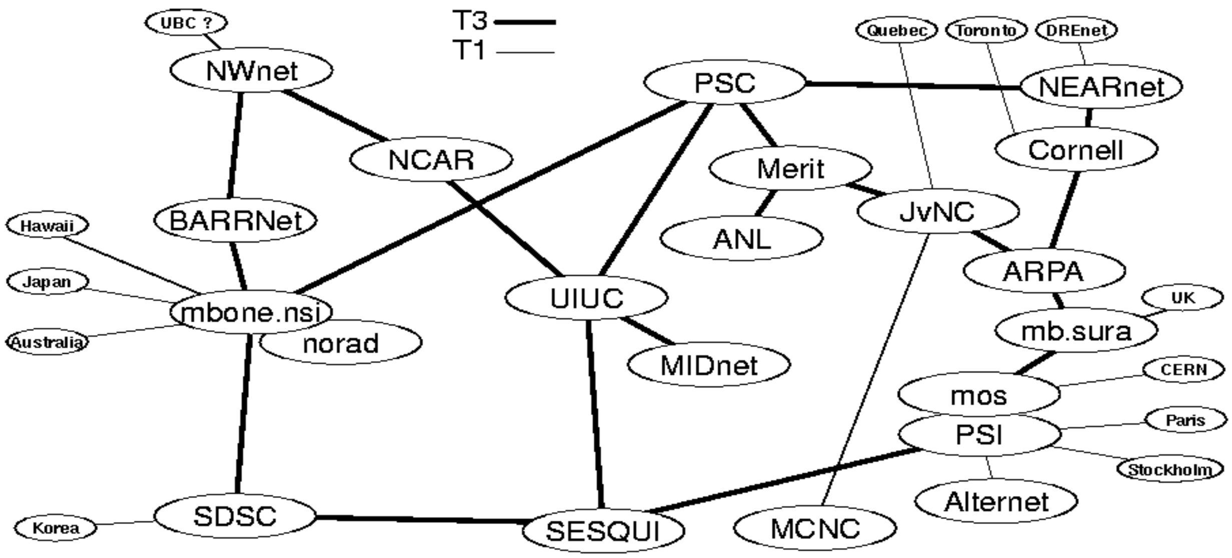
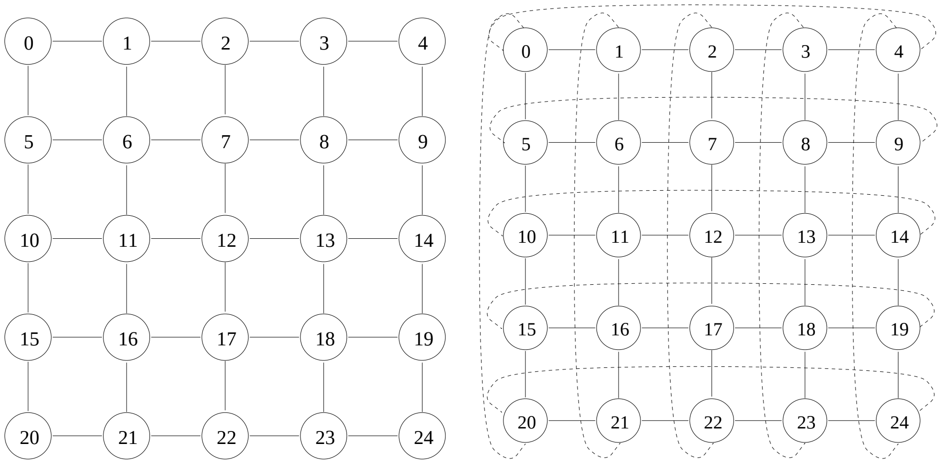


Figure 6: The proposed MBONE topology in North America

the links connecting nodes in all three topologies consist of two unidirectional links (running in opposite directions), each with 100 Mbps transmission bandwidth. Among these three, the wrapped rectangular mesh is the best in terms of connectivity, and the unwrapped one is the second.

We established real-time channels over these three networks. For the sake of simplicity, we used real-time channels with identical input traffic specification and performance requirements. Thus, the real-time channels to be established are specified by (C_{max}, p, d)=(3 msec, 33 msec, 100 msec). According to the admission test described in Section 4, up to 10 real-time channels can be established over each link. In order to evaluate the probability of successful channel establishment, we have set load conditions as follows. First, we selected source–destination pairs randomly, and established a real-time channel between each pair using the proposed scheme. The lifetimes of these channels are set infinite, and thus, the number of channels monotonically increases with time. After a certain amount of time elapsed, we counted the number of real-time channels established over each link, added them up, and divided the sum by the number of links of the network. The resulting number was treated as



(a) Ordinary mesh (b) Wrapped mesh

Figure 7: Network topologies used for the simulation

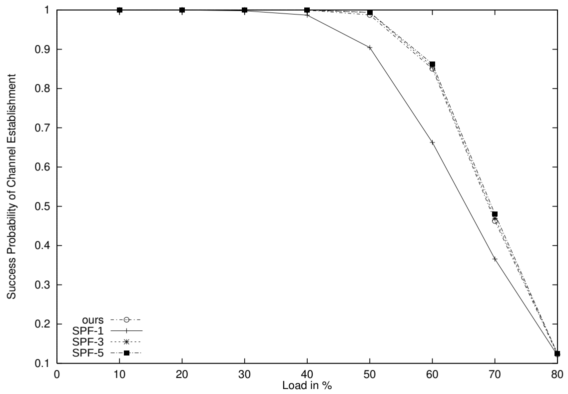
the average load of the network. Under this setting, we (i) generated channel-establishment requests whose inter-arrival times are exponentially-distributed, (ii) executed the proposed scheme, and (iii) measured the success probability. In order to calculate the success probability under a constant load condition, we disconnected the newly-established channels in a very short time. Without this process, the load will not be constant but increase continually because of the new channels added. For the purpose of comparison, we applied the SPF algorithm to the same source–destination pairs and measured the success probability. That is, we first find the minimum-hop route between source and destination, then send a channel-establishment request along the chosen route. If the channel-admission tests succeed at every node along the path, the destination sends a channel-acceptance message backward to the source. Then, the source considers the establishment successful. If channel-admission test fails at a particular link, we consider the link disconnected in the topology and initiate another search for a new minimum-hop route and send a request message along the new path. We repeat this process up to 5 times until the channel establishment succeeds.

During this experiment, we also counted the number of messages needed to establish a real-time channel using both approaches in order to evaluate their operational overheads. Finally, we measured the time to establish each channel for both approaches.

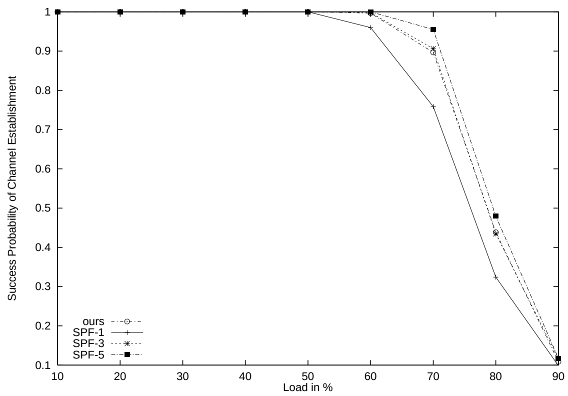
The above process was executed while varying the given network load and the entire procedure was executed 100 times.

5.2 The Simulation Results

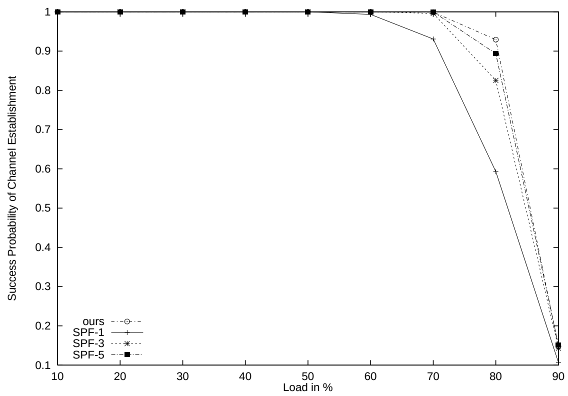
Fig. 8(a) shows the establishment success rate under various loads in the MBONE topology using both our approach and the sequential search based on SPF. SPF-n denotes a scheme that allows up to n trials of SPF. In Fig. 8(a), when the network is lightly-loaded (i.e., the network load is under 30%), both approaches achieve an almost 100% channel-establishment success rate. That is, abundant resources are available in such a case and hence, most of channel-establishment requests are accepted. When the network load is in the mid-range (i.e., 40-70%), the success rate of our scheme is comparable to that of SPF-3. In fact, SPF-5 shows little improvement over SPF-3. Compared to SPF-1, our scheme shows a much higher success rate in this range. Ideally, our approach must outperform any sequential search method as our parallel search is guaranteed to find a qualified route, if any. In practice, however, due to the reservation of link bandwidths by pending real-time channels which had been requested almost simultaneously, its rejection rate of channel-establishment requests is a little higher as seen in Fig. 8(a). When the network is heavily-loaded (e.g., over 80%), our approach and the sequential approaches, irrespective of the number of trials, both show low success ratios. This is due mainly to the limited connectivity of the MBONE network topology. In fact, we could not obtain any meaningful results when the network load is 90%, because the number of real-time channels established was very small.



(a) MBONE



(b) Ordinary mesh



(c) Wrapped mesh

Figure 8: Channel-establishment success rate

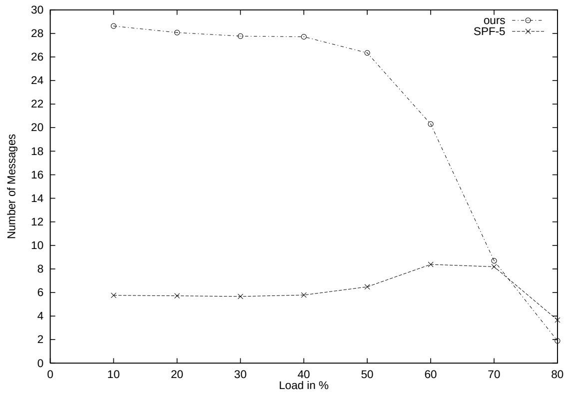
Fig. 8(b) shows the establishment success rate for the unwrapped mesh. Compared to the

MBONE topology, the success rate is higher. In particular, our scheme's success rate is comparable to that of SPF-3. Fig. 8(c) shows the establishment success rate for the wrapped rectangular mesh. Compared to the unwrapped mesh and the MBONE, it shows much higher success rate. In particular, our scheme's success rate is shown to have a larger increase than SPFs; in fact, our scheme shows the highest success rate under most loads. This clearly demonstrates the superiority of our scheme for networks with better connectivity.

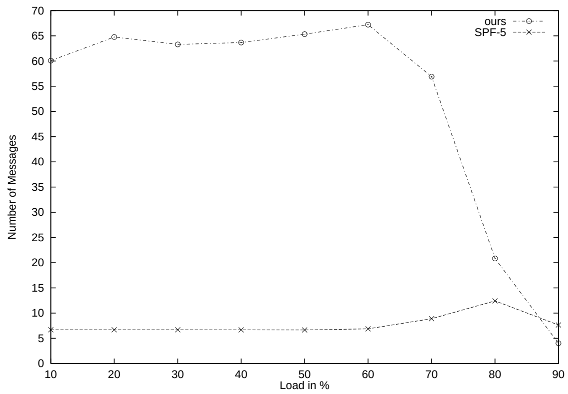
In order to compare the operational overheads of our scheme and the sequential SPF, we measured the number of establishment-request messages sent over each link plus the number of channel-acceptance messages, and averaged them over the entire network. Under lightly-loaded conditions, the number of messages sent in our scheme is very large: approximately, 28, 65 and 85 in the MBONE, the unwrapped mesh and the wrapped mesh, respectively. This results from the fact that our scheme employs a form of limited flooding and under a lightly-loaded condition, most links can accommodate a new establishment request and relay request messages to their neighbor nodes. Moreover, it is often the case that multiple copies of the same request pass through a link. It is important to note, however, that this larger message overhead is "harmless," as the establishment request messages use idle resources (likely to exist in lightly-loaded networks).

As the network gets more loaded, the operational overhead decreases, demonstrating that the pruning effect of our scheme works well as the network load gets higher. That is, as the network load gets higher, some links reject a new establishment request due to their insufficient available bandwidth, and hence, do not propagate the request to their neighbors, thus decreasing the number of request messages. In particular, when the load is 90%, the number of messages was approximately 5 in the unwrapped mesh; the average number of hops of newly-accepted real-time channels is approximately 1. This also confirms that our scheme can keep the operational overhead under a highly-congested condition low enough to save link bandwidth for other best-effort traffic.

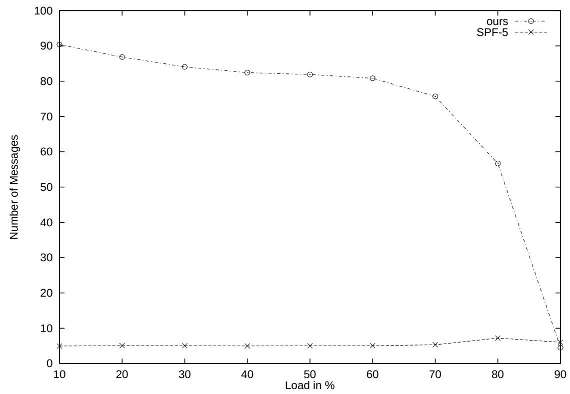
Fig. 9 shows the operational overheads for our scheme and SPF-5 for all three topologies. Compared to our scheme, SPF-5 generates a considerably smaller number of request messages, because under lightly-loaded conditions, the first establishment trial succeeds in most cases. While the operational overhead of our scheme decreases with the load, the overhead of SPF-5 increases in the mid-range load. This demonstrates that, as the network load gets heavier, the success probability of the first trial decreases and the second & third trials are required, thus increasing the number of request messages. Under an extremely-congested condition (e.g., 80% load for MBONE and 90% load for unwrapped and wrapped meshes), the operational overhead of SPF-5 decreases again for a reason similar to what happened to our scheme. Very few request messages propagate through the network, because there aren't many links with sufficient available bandwidth. Interestingly, our scheme generated smaller overhead than SPF-5 in this case, because we included both failure-notification and channel-acceptance messages in the calculation of overhead. Unlike our scheme, SPF-5 requires a node which cannot forward the request to the next node because of insufficient network resources to send a failure-notification message back to the source.



(a) MBONE



(b) Unwrapped mesh

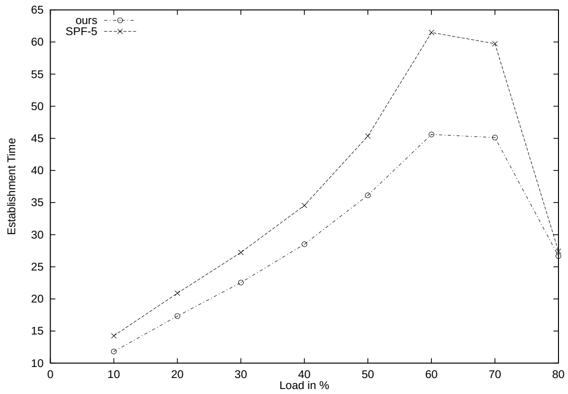


(c) Wrapped mesh

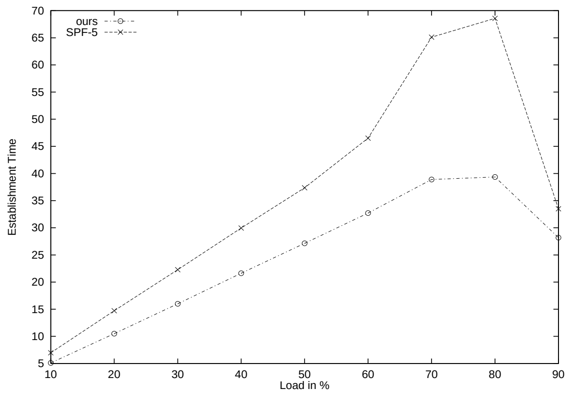
Figure 9: Comparison of overheads

The overhead analysis in Fig. 9 may indicate our scheme to be disadvantageous compared to the sequential search, but it isn't so for two reasons. First, the large number of request messages in a lightly-loaded network doesn't do any harm; since the links are not used often for transmitting normal traffic when the network is lightly-loaded, request messages can be transmitted without significantly delaying or dropping regular packets. Considering the existence of non-real-time packets, we need to introduce an appropriate measure so that the number of channel-establishment request messages may be reduced. This can be achieved by limiting the number of hops that the request messages can be relayed from the source, as mentioned earlier. Second, our scheme takes much less time in establishing a real-time channel than the sequential search. The *channel-establishment time* is defined as the time measured from the issuance of a request to the notification of its acceptance. Fig. 10 shows the channel-establishment times of both our scheme and SPF-5, where the time unit is a packet-transmission time over a single link. We assumed that the packet size is constant throughout the network. In this figure we showed the establishment time for only successful requests. As can be seen in Fig. 10, our scheme is considerably better than the SPF-5.

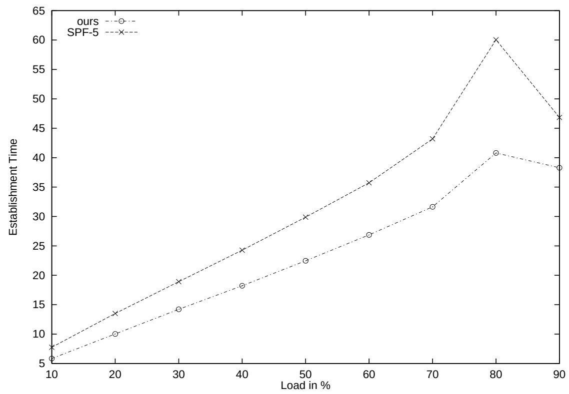
In addition to its larger channel-establishment time, the sequential search based on SPF requires



(a) MBONE



(b) Unwrapped mesh



(c) Wrapped mesh

Figure 10: Comparison of channel-establishment times

to find the shortest path for each trial using an appropriate algorithm such as Dijkstra's [2], which is itself time-consuming. By contrast, our scheme does not require execution of such an algorithm. In order to assess the effect of SPF algorithm execution on the channel-establishment time, let's consider the following simple example, and assume that Fig. 7 (b) is an ATM network, each link with 155 Mbps bandwidth, and that the size of a channel-establishment request message is the same as a single ATM cell (53 bytes). Since the information which must be carried in a request message is minimal as discussed in Section 3, this assumption is justifiable. In this network, the transmission time of a request is approximately 2.735×10^{-6}. When a benchmark test of the SPF algorithm used in our simulation was performed on SUN SPARC 20, the average execution time was approximately 1.7 msec when the source and destination pairs were randomly generated. Considering only this fact, one can easily observe the large overhead of executing the SPF algorithm. Specifically, in the unwrapped rectangular mesh, the average number of trials of SPF-3 whose success rate is comparable to that of our scheme is approximately 1.4 in the case of 70 % load, and thus, the execution time of the SPF algorithm is 2.36 msec[3]. Compared to the transmission time of request messages, this

[3] One may employ a k-shortest-path-finding algorithm [18] to save time in calculating shortest paths in each trial.

execution time is an order of magnitude larger, demonstrating the superiority of our scheme to the sequential SPF algorithm. Moreover, by eliminating the need for an SPF routine, our approach becomes very attractive to embedded real-time systems with low-cost, low-power CPUs.

6 Conclusion

In this paper, we have proposed and evaluated an efficient distributed route-selection scheme which is guaranteed to find a qualified route, if any, for each real-time channel-establishment request. By equipping two simple tables with each node, the proposed scheme can not only eliminate the common reliability and performance bottlenecks of centralized route selection, but also keep the operational overhead sufficiently low for practical use.

The proposed scheme is presented in procedure form, and its correctness and completeness are discussed. Several examples are presented to illustrate how the proposed scheme works. Our simulation results have also shown this scheme to make significant performance improvements over the sequential search based on SPF, especially in terms of channel-establishment time at a reasonable cost.

Appendix: Additional Examples

Before proceeding to the examples, we first describe an environment in which such examples will be derived. The same network in Example 1 will be used for Example 2 (Fig. 11), but TECs are no longer empty, i.e., there already exist many real-time channels when a new channel-establishment request is made. Table 2 shows the content of all TECs at some time instant. The entries of these tables had been inserted for establishing the following 21 channels. Each channel is described as a 5-tuple $(ID, S_{max}, p, D, path)$. Table 3 presents the 21 channels in the order of their establishment. Note that the TECs in Table 2 are only one of many *possible* situations after these 21 channel-establishment requests have been processed and accepted. As discussed earlier, due to the randomness of network traffic, there may be many other possible sets of TECs. We also illustrate such situations in the following examples.

Example 2: We want to establish a real-time channel with ID =1:8, $S_{max} = 200Kb$ and $p = 20\,ms$ in the network of Fig. 11 under the environment as specified in Table 2. The source of the requested channel is N1 and its destination is N5. We will first let $D = 19$ to show that only the path N1N2N4N5 is a qualified route for this request. D will then be changed so as to illustrate other cases.

However, the success ratio of this approach is lower than the one used in this simulation since the identical topology is used in calculating all the shortest paths while the serial search used in this simulation removed the disconnected link during each trial.

ID	C	p	d	ID	C	p	d	ID	C	p	d	ID	C	p	d
TEC: 1 → 2				TEC: 2 → 1				TEC: 1 → 3				TEC: 3 → 1			
3:4	1.0	10	4.5	2:4	1.0	10	4.5	2:4	2.0	10	5.5	3:4	2.0	10	5.5
1:6	1.0	20	6	5:3	1.0	20	6	1:3	1.0	20	8	5:2	1.0	20	8
1:4	0.8	25	9.6	4:1	0.4	25	9.8	1:2	0.6	20	16.2	3:3	2.0	20	20
1:7	2.0	10	10	2:3	2.0	10	10	1:5	2.0	20	20	4:2	1.6	50	23.8
1:1	0.5	50	33	2:2	1.0	40	19.5	2:2	2.0	40	20.5				
TEC: 3 → 4				TEC: 4 → 3				TEC: 2 → 4				TEC: 4 → 2			
1:2	1.5	20	17.1	5:4	3.0	20	10.9	1:6	2.0	20	7	5:3	2.0	20	7
3:1	1.0	40	20.3	5:1	3.0	20	15.9	1:4	1.6	25	10.4	4:1	0.8	25	10.2
3:2	5.0	40	21.5	4:2	4.0	50	26.2	1:1	1.0	50	33.5	3:2	2.0	40	18.5
								2:1	2.0	50	50				
TEC: 3 → 5				TEC: 5 → 3				TEC: 4 → 5				TEC: 5 → 4			
1:3	5.0	20	12	5:2	5.0	20	12	1:6	2.0	20	7	5:3	2.0	20	7
								3:1	0.4	40	19.7	5:4	1.2	20	9.1
								1:1	1.0	50	33.5	5:1	1.2	20	14.1
								2:1	2.0	50	50				

Table 2: TECs for Example 2

ID	S_{max}	p	D	path	ID	S_{max}	p	D	path	ID	S_{max}	p	D	path
1:1	50	50	100	N1N2N4N5	1:2	30	20	33.3	N1N3N4	3:1	20	40	40	N3N4N5
2:1	100	50	100	N2N4N5	5:1	60	20	30	N5N4N3	4:1	40	25	20	N4N2N1
4:2	80	50	50	N4N3N1	2:2	100	40	40	N2N1N3	1:3	50	20	20	N1N3N5
5:2	50	20	20	N5N3N1	3:2	100	40	40	N3N4N2	1:4	80	25	20	N1N2N4
3:3	100	20	20	N3N1	1:5	100	20	20	N1N3	1:6	100	20	20	N1N2N4N5
5:3	100	20	20	N5N4N2N1	1:7	200	10	10	N1N2	2:3	200	10	10	N2N1
2:4	100	10	10	N2N1N3	3:4	100	10	10	N3N1N2	5:4	60	20	20	N5N4N3

Table 3: The table of previously-accepted requests for Example 2.

N1's operations: The MWRTs (r values in TPR) and C values of N1's outgoing links are computed first. For link $1 \rightarrow 3$, $C = 200/50 = 4$. To compute a channel's MWRT on a link, we need to assign highest possible priority to this channel without violating the link deadlines (d in TEC:$1 \rightarrow 3$) 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. Recall that the existing channels are placed in a TEC in ascending order of their d values. By giving the newly-requested channel top priority on link $1 \rightarrow 3$, the worst-case delay of channel 2:4 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 2:4 but above channel 1:3. With this 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 [11–13]:

$$\exists t \in B_i = \{d_i\} \cup \{kp_j : \ j \in A_i, 0 \leq k \leq \lfloor \frac{d_i}{p_j} \rfloor \},$$
$$W_i(t) = \sum_{j \in A_i} C_j \cdot \lceil \frac{t}{p_j} \rceil + C_i \leq t,$$

where A_i is the set of channels (including the requested channel) whose priority is higher than that of channel i, C_i is the maximum service time of channel i, d_i is the link deadline of channel i, and p_i is the minimum message inter-arrival time of channel i. Given below is an acceptable set of t and $W_i(t)$ for channels on link $1 \rightarrow 3$.

- channel 1:3, $t = d_{1:3} = 8$, $W_{1:3}(8) = 7 < 8$.

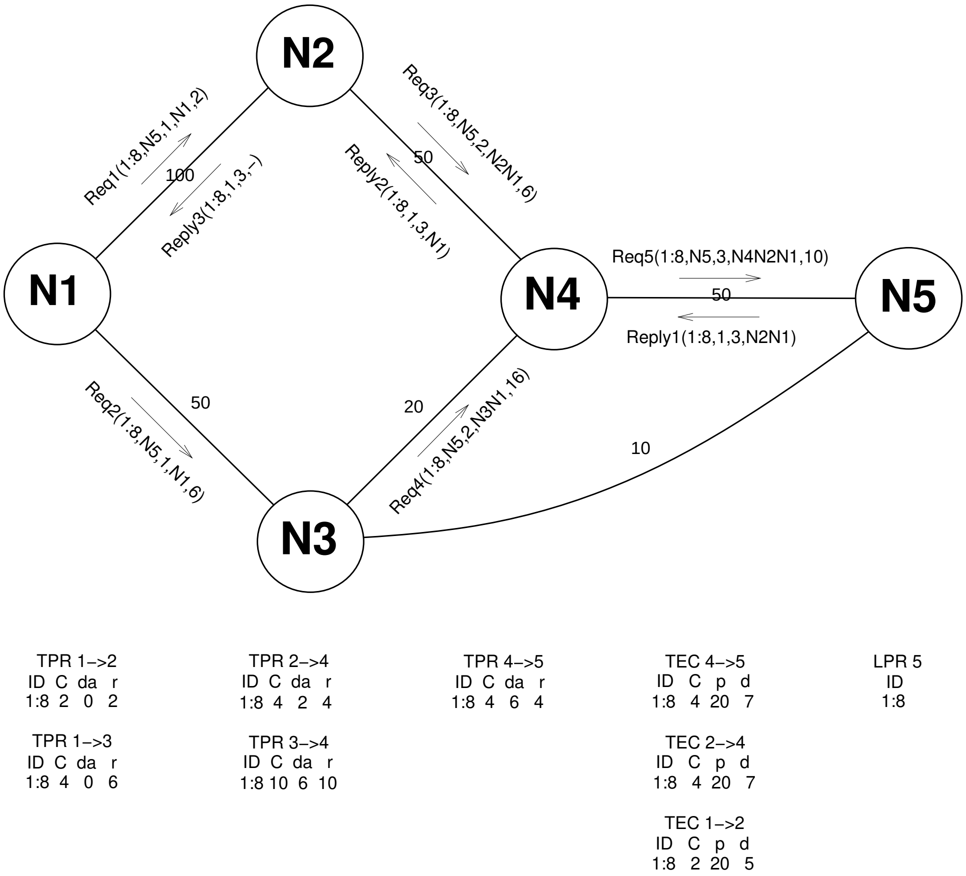


Figure 11: Example 2

- channel 1:2, $t = d_{1:2} = 16.2$, $W_{1:2}(16.2) = 9.6 < 16.2$.
- channel 1:5, $t = d_{1:5} = 20$, $W_{1:5}(20) = 11.6 < 20$.
- channel 2:2, $t = p_{1:3} = 20$. $W_{2:2}(20) = 13.6 < 20.5$.

As a result, the priority of the requested channel will be placed between channel 2:4 and channel 1:3. So, the MWRT of the requested channel on link $1 \rightarrow 3$ is equal to the smallest t such that $W_{1:8}(t) = t$, i.e., the smallest t such that $2 \times \lceil \frac{t}{10} \rceil + 4 = t$. This channel's MWRT on link $1 \rightarrow 3$ is therefore 6. For link $1 \rightarrow 2$, using the same procedure, the MWRT is computed to be 2 by giving the new channel the highest priority on this link.

Note that this priority assignment for the requested channel is just for computing the MWRT. As discussed in Section 3.1 and in [11, 12], the run-time priority in transmitting the messages of the requested channel, if accepted, is determined based on the channel's link-delay deadline $\geq$ the MWRT obtained here and will be determined after the route is selected.

After computing MWRTs, N1 inserts appropriate entries into TPR:$1 \rightarrow 2$ and TPR:$1 \rightarrow 3$ (as shown in Fig. 11). Channel-establishment requests are sent to both N2 and N3. Fig. 11 shows only 5 fields in the request messages as in Example 1, because the other fields are the same for all request messages.

Operations of N2 and N3: After receiving the request message from N1 and determining that this request is new (does not exist in any TPR and TEC), N2 and N3 will perform similar operations as N1, i.e., compute C and MWRT values for all of their outgoing links except for the links to N1, and store this information in TPRs.

Using the same procedure as above, we get $C = 4$ and MWRT$= 4$ on link $2 \rightarrow 4$ by assigning the new channel top priority. A request message (Req3 in Fig. 11) is sent to N4 with $d^a = 2 + 4 = 6$.

For link $3 \rightarrow 4$, we get $C = 10$ and MWRT$= 10$ by assigning the new channel top priority. A request message (Req4 in Fig. 11) is sent to N4 with $d^a = 6 + 10 = 16$.

For link $3 \rightarrow 5$, we get $C = 20$ and MWRT$= 25$ by assigning the new channel the lowest priority. However, in order to make the scheme work correctly, we require $d \leq p$ $(= 20)$ on each link of the channel's route [11, 12]. So, this link cannot be part of a qualified route, and hence, no request message is sent to N5 via this link and no entry (corresponding to this request) will be inserted into TPR:$3 \rightarrow 5$.

N4's operations: Since it is not certain which of Req3 or Req4 will arrive at N4 first, we will discuss both cases.

If Req3 arrives first, then N4 will compute C and MWRT for links to both N3 and N5. For link $4 \rightarrow 3$, we get $C = 10$ and MWRT$= 16$ by assigning the requested channel the priority lower than channel 5:1 but higher than channel 4:2. So, the accumulated MWRT from N1N2N4N3 is $6 + 16 = 22 > D(= 19)$, and hence, no request message will be sent to N3 and no entry (corresponding to this request) will be added to TPR:$4 \rightarrow 3$.

For link $4 \rightarrow 5$, we get $C = 4$ and MWRT$= 4$ by assigning top priority to the requested channel. So, N4 stores this information in TPR:$4 \rightarrow 5$ and Req5 (with $d^a = 6 + 4 = 10$) is sent to N5. After N4 receives Req4, since the d^a (=16) carried in the message is greater than that (=10) in N4's TPRs, Req4 is discarded.

On the other hand, if Req4 arrives at N4 first, N4 will compute C and MWRT for links to N2 and N5 based on Req4. For link $4 \rightarrow 5$, because MWRT$= 4$, the accumulated MWRT, d^a, from N1 to N5 will be $16 + 4 = 20$ which is greater than the required end-to-end delay bound 19, so no message will be sent to N5. A similar situation occurs for link $4 \rightarrow 2$, since $C = 4$ and MWRT$= 4$ (top priority). Thus, when Req3 arrives, the operations performed by N4 will be exactly the same as those in the case when Req3 arrives at N4 first.

Reply operations: N5 will perform the same operations as in Example 1, but with $diff = (19 - 10)/3 = 3$. The operations to be performed by N4, N2 and N1 when the reply message arrives at

	MWRT		
Path	CH 1:8 first	Same time	CH 2:5 first
N1N2N4N5	10	25.2	25.2
N1N3N4N5	20	27.6	36.6
N2N4N3	NA	20	20
N2N1N3	17	17	8

Table 4: MWRT for two concurrent requests with $S_{max} = 200Kb$.

these nodes are the same as in Example 1.

If we increase D to 20, the path N1N3N4N5 with the worst-case accumulated delay 20 is a qualified route for the requested channel. Thus, if Req4 arrives at N4 before Req3 (Fig. 11), this route will be chosen instead of N1N2N4N5.

On the other hand, if we let $D = 15$ in this example, as can be seen in Fig. 11, Req4 will not be sent because its $d^a > 15$. If we decrease D further to 9, then Req5 will not be sent. In this case, no qualified route exists at that time, so the channel-establishment request will time out and thus, will be rejected. □

Example 3: In this example, we will use the same network and environment in Example 2 to demonstrate the effects of over-estimating link delays in case of multiple pending requests. In addition to the requested channel 1:8 in Example 2, we assume another channel (ID =2:5) establishment request occurs at about the same time. This channel is specified as $S_{max} = 200Kb$, $p = 20\,ms$, source = N2 and destination = N3. At first, we ignore the end-to-end delay requirements (D) of both channels, i.e., assume both Ds are sufficiently large.

If these two channel requests arrive sequentially with a sufficiently large inter-arrival time between them (regardless of the order of their arrival), the least MWRT of channel 1:8 is equal to 10 (via path N1N2N4N5), and that of channel 2:5 is equal to 8 (via path N2N1N3). Note that as discussed in Section 3, due to the randomness of the network traffic, the path with the least MWRT may not always be chosen. So, the MWRT of a channel depends on which path is chosen. Thus, the least MWRT of a channel is defined to be the MWRT of the path with the smallest MWRT. However, when the inter-arrival time between them is not sufficiently large, i.e., TPRs contain entries for both channels at the same time, the MWRTs (including the least MWRT) may be over-estimated. Table 4 shows the MWRTs of the two requests via two (with the least and the second least MWRT) of their possible routes for different arrival orders of channels 1:8 and 2:5. The first column shows the MWRT when channel 1:8 arrives first and channel 2:5 arrives before the deletion of channel 1:8's entries (in TPRs). The third column shows the opposite, i.e., channel 2:5 arrives first. The second column represents the situation when channel 1:8 request arrives at N1 approximately at the same time as channel 2:5 request arrives at N2. Note that the second column is derived under the assumption

	MWRT		
Path	CH 1:8 first	Same time	CH 2:5 first
N1N2N4N5	2.5	4.5	4.5
N1N3N4N5	4.5	5.5	6.5
N2N4N3	7	3.5	3.5
N2N1N3	2.5	2.5	1.5

Table 5: MWRTs for two concurrent requests with $S_{max} = 50Kb$.

that request messages travel faster via links with smaller MWRTs and the first-come-first-serve link scheduling policy for messages of the same priority.

As can be seen from the first and third columns of Table 4, the MWRT estimate for the channel which arrives later is unnecessarily large. When two channels arrive at about the same time, the MWRT estimates for both channels are larger than their true value. Note, however, we have not yet considered the end-to-end delay requirement. When D is reasonably large, e.g., $D_{1:8} \geq 26$ and $D_{2:5} \geq 17$, qualified routes for both channels can be found regardless of the over-estimation of link delays. In fact, most real-time applications do not require such short end-to-end delays (at least 100 ms for typical interactive applications) and do not generate such high-volume data streams. For example, if we decrease S_{max} to $50Kb$ which is a typical size for multimedia applications, the least MWRT for channel 1:8 is 2.5 ms, and for channel 2:5 is 1.5 ms. Table 5 shows the MWRTs of the same four paths (as in Table 4) in this case. As can be seen from Table 5, channel 1:8's MWRT increases only by 2 ms in the worst case due to the over-estimation of link delays and channel 2:5's MWRT only by 1 ms. This example shows that, unless the requested channel generates very large messages or it requires a very short end-to-end delay, over-estimation usually does not cause unnecessary denial of channel-establishment requests. $\square$

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, Englewood Cliffs, New Jersey, second edition, 1992.

[3] 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*, pp. 86–96, February 1994.

[4] C.-C. Chou and K. G. Shin, "Statistical real-time channels on multiaccess networks," *Proc. The International Conference on Computer Communication*, pp. 29–34, August 1995.

[5] C.-C. Chou and K. G. Shin, "Multiplexing statistical real-time channels on a multiaccess network," *Proc. 16-th Int'l. Conf. on Distributed Computing Systems*, pp. 133–140, May 1996.

[6] D. E. Comer, *Internetworking with TCP/IP Volume I, Principles, Protocols, and Architecture*, Prentice-Hall International, Englewood Cliffs, New Jersey, third edition, 1995.

[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] S. Daniel, K. G. Shin, and S. Yun, "A router architecture for flexible routing and switching in point-to-point networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 10, no. 1, pp. 62–75, January 1999.

[9] 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.

[10] 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.

[11] 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. (An expanded version appeared in *IEEE Trans. on Parallel and Distributed Systems*, vol. 5, no. 10, pp. 1044–1056, Oct. 1994.).

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

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

[14] PNNI Working Group. *ATM Forum 94-0471R13 PNNI Draft Specification.* Document available at ftp://ftp.atmforum.com/pub/contributions.

[15] 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.

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

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

[18] A. Weintraub, "The shortest and the K-shortest routes as assignment problems," *Networks*, vol. 3, pp. 61–73, 1973.

[19] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: a new resource ReSerVation protocol," *IEEE Network*, pp. 8–18, September 1993.