24

A Distributed Route-Selection Scheme for Establishing Real-Time Channels

Kang G. Shin and Chih-Che Chou

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

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 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. Simulation results are presented to demonstrate the effectiveness of the proposed distributed route-selection scheme.

1 Introduction and Problem Statement

The increasing demand of real-time network services has generated considerable interest in the development of real-time communication protocols. The concept of "real-time channel" proposed by Ferrari and Verma [4] is one of the most notable solutions to the problem of meeting message/packet[1] delivery deadlines in wide-area point-to-point networks. 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 [4].

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

The work described in this paper was supported in part by the Office of Naval Research under Grant N00014-94-1-0229 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] We will use the term "message" throughout the paper, but it could be replaced by the term "packet", depending on the underlying context.

R. Puigjaner (ed.), *High Performance Networking*

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 [2,4,5,7], very few of them have addressed explicitly the issue of selecting a route between the source and destination of a channel, 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, 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 basically two approaches to the route-selection problem: centralized or distributed. Most existing channel-establishment schemes are based on the centralized approach [5,7].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, 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. Although one can devise efficient algorithms for the network manager to select qualified routes, the centralized approach suffers both performance and reliability bottlenecks due to the use of the network manager.

In contrast with the centralized approach, the distributed route-selection approach can avoid performance and reliability bottlenecks. However, choosing a qualified route among all possible routes between the source and destination may not be an easy task, because there could be many possible routes between two communicating peers. To guarantee the discovery of a qualified route, if any, we have to search all possible routes between the source and destination of a channel to be established, while keeping the operational overhead low enough to make the scheme practically feasible. In this paper, we propose an efficient distributed route-selection scheme that satisfies the above requirement for a single establishment request at a time. The scheme also works well for multiple simultaneous requests if the existing real-time traffic is reasonably low.

The paper is organized as follows. Our proposed solution to this problem and its overhead analysis are presented in Section 2. In Sections 3, simulation results are presented to show the effectiveness of the proposed solution. The paper concludes with Section 4.

2 The Proposed Solution Approach

We first describe the environment and the assumption under which our distributed route-selection scheme is developed. The underlying network is an arbitrary point-to-point network. As in [3,5,8], 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, there are two restrictions on each arrival: (1) the number of messages generated in any time interval of length t does not exceed $B_{max} + tR_{max}$; (2) the length of each message does not exceed S_{max}. Based on this message arrival model, the authors of [5] 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 real-time channel-establishment request. Using the delay-estimation method in [5] and a Bellman-Ford-like algorithm, we will in this paper develop a scheme to find a qualified route for each channel-establishment request.

2.1 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.

The goal of the algorithm in [5] 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, $p_i = I^i_{min} = \frac{1}{R^i_{max}}$ 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 [5] to work correctly. Given a new channel $M_{k+1} = (C_{k+1}, p_{k+1})$ to be established, the authors of [5] 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. 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, p and d (over link ℓ) for all existing channels are available.

The method in [5] has *not* included 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. 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. This over-estimation of MWRT may sometimes 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 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.

2.2 The Route-Selection Algorithm

Based on the above definition of link delay, we can apply the Bellman-Ford algorithm [1] to solve the route-selection problem. However, under the original Bellman-Ford algorithm, only the one which has the shortest delay is explored at any time, while our algorithm simultaneously explores *all* routes which are possible to be a shortest path.

Each node has to maintain two sets of tables to store the characteristics information 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: (1) 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. (2) The maximum service time of a message (C) of this channel. (3) The minimum message inter-arrival time (p) of this channel. (4) 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 [5], 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.

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 and consists of six fields. The first three fields are the same as those of a TEC and the remaining three fields are: (1) d^a: the accumulated delay from the source to the current node, (2) *timeout*: the expiration time of this request message, (3) r: MWRT of the corresponding outgoing link.

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. Then 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. Copies of this channel-establishment message will be put into the queues of all of its outgoing links at the same time, 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.

```
Procedure rcv_req
If (Req.timeout ≤ current_time) then discard_req;
else if (Req.ID ∈ TEC) then discard_req;
else if (Req.destination = A) then reply_req;
else if (Req.ID ∈ TPR) then {
        if (Req.d^a ≥ 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 of processing a channel-establishment request

Fig. 1 outlines the procedure an intermediate node A will execute when a real-time channel-establishment request is received. It checks the received request message to determine whether the message should be discarded or processed further. The first two **if** statements discard requests which are expired or already accepted. Procedure *reply_req* will then be called in if node A is the destination of the channel.

The fourth **if** clause is for the requests already in TPRs. The request will be discarded if d^a of the received message is not smaller than the corresponding one in TPRs. Otherwise 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. Finally, we conclude that the request is new to node A. Thus, the request message will be appropriately stored and forwarded if the sum of the accumulated delay and the MWRT of the corresponding next link is less than D. Fig. 2 shows the procedures of inserting a new channel-establishment request and forwarding a request.

Each destination node has to keep a temporary list of already-processed requests (LPRs) in order to avoid reporting a 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 channel-establishment request. From Procedure *reply_req*, one can see that if the system decides to accept the channel-establishment request, the (qualified) path carried by the request arrived first will be selected as the route for the channel.

If d^a (the sum of MWRTs of all links on the path from the source to destination) is smaller than the user-specified end-to-end delay bound D, 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 into *hops* parts at the destination and

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 the immediate upstream node.

Figure 2: Procedures of inserting and forwarding a request

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 of processing a channel-establishment request at the destination

distributed to all links along the path [5]. 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 $(D-d^a)/hops$. Thus, $(D-d^a)/hops$ is included in the channel-establishment confirmation message (by *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 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 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

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 of handling reply messages

be included. All the intermediate nodes will delete the corresponding entries in their TECs upon receiving the disconnection message.

2.3 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) [1] algorithm can ensure the least MWRT path to be found, although other larger-delay routes may be found first. Thus, 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 (if it is "qualified"), 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.

Another performance measure is the time needed for establishing 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 destination then back to the source node 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 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 establish-

ment request. Note that "one time (copy)" is defined as "sending a message across one link". For a more accurate estimation, the request message is assumed to travel faster through a lightly-loaded link (while 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, we may find that each node will 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 also likely receive the copy of a request message which travels through the route with the smallest value of $Req.d^a$ first, and thus, all subsequent copies of the same request message will be discarded. Under this assumption, a request message will therefore be transmitted at most $2K$ times in the worst 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.

The conditional statement ($Req.d^a + r < Req.D$) in Fig. 1 is used to stop the unnecessary propagation of a request to a region where no qualified routes exist. 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; i.e., stop forwarding a request if the number of hops the message has traveled so far is more than a pre-defined limit. If the pre-defined limit can be chosen appropriately, this method may 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.

3 Simulation Results

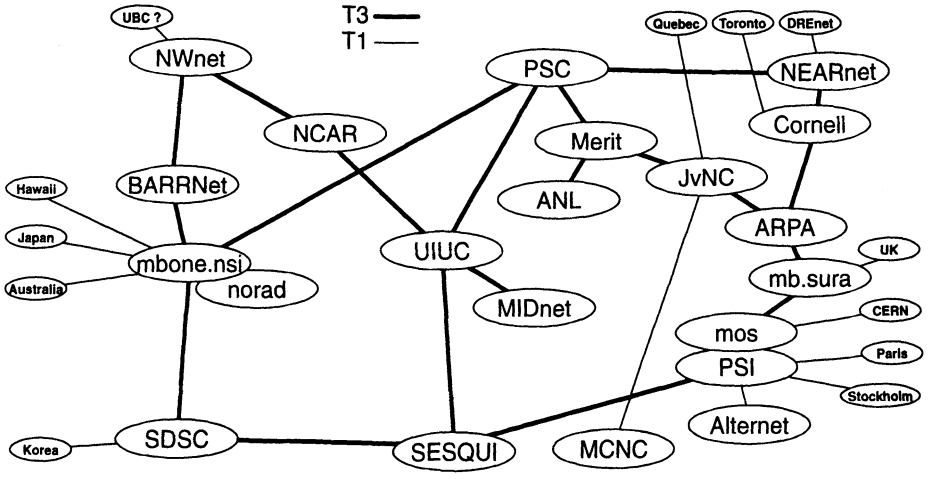

Figure 5: The proposed MBONE topology in North America

In this section, we present a numerical example to demonstrate the effectiveness of our route-selection strategy for real-time channels. We use a part of the proposed MBONE network in the North America region (Fig. 5) for transmitting digital video frames. All T3 links and nodes which are connected via T3 links are included in our simulation. The video data (figure omitted due to space limitation) used are obtained from a 5376-frame (for about 3 minutes at the rate of 30 frames per second) sequence of the movie "Star Wars" [6]. The maximum one-way transmission delay of a frame is assumed to be less than 100 ms (including only the queuing delay and the actual transmission time) in order to achieve the quality of live video. At the transmission rate of 30 frames per second, 3 frames will be transmitted in each 100 ms. The maximum packet size of the network is assumed to be 1 Kbytes. Based on this model, we simulate and compare the proposed strategy with and without the "hops" limit and also the conventional shortest-path route-selection strategy.

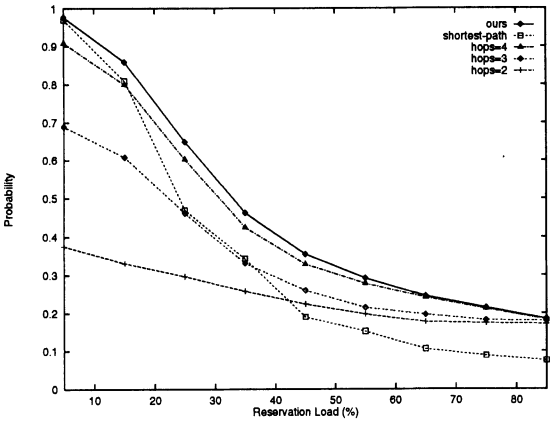

Figure 6: The success probability of channel-establishment requests

Fig. 6 shows the success probability of real-time channel-establishment requests under different network load conditions. The "hops=n" in Fig. 6 represents our strategy whose hop limit equals n. As can be seen from the figure, the performance of the shortest-path strategy is sensitive to the change of load condition because once the shortest-path is congested, the shortest-path strategy cannot find an alternate route. When the network is heavily-loaded, all lines except for the shortest-path line converge to the same point, since at a such high network load, most channel-establishment requests with distance greater than 2 hops will be rejected. Therefore, the hop limit becomes immaterial in this case. However, when the network is lightly-loaded, the strategy with a higher hop limit outperform those with a lower hop limit by a significant margin.

Fig. 7 shows the average number of copies of messages per channel-establishment request (including the confirmation messages). As expected, a strategy with a higher hop limit generates more copies of messages, and yields a higher success probability. However, the difference becomes smaller and smaller when the network load increases. Since when the

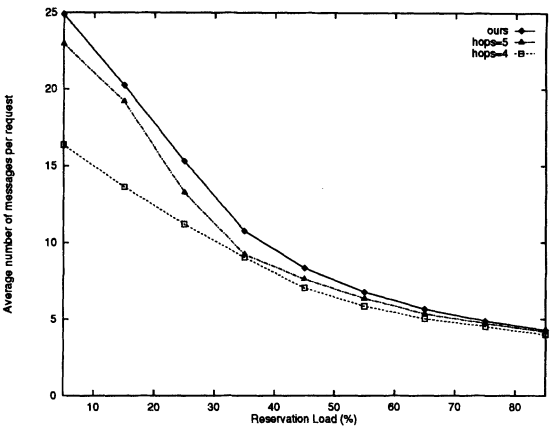

Figure 7: The average number of messages per request

network becomes heavily-loaded, our strategy with a higher hop limit will automatically stop sending messages along longer routes as soon as it discovers that these routes cannot possibly provide the required end-to-end delay bound even before the hop limit is reached.

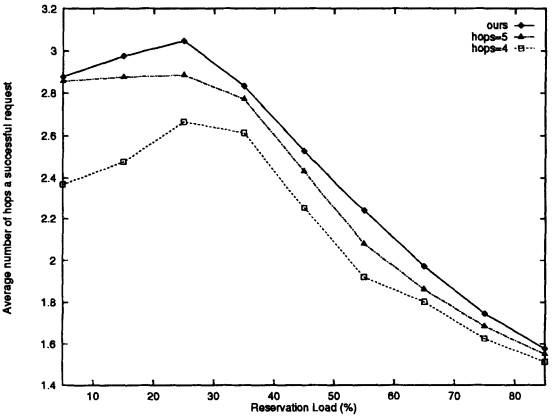

Figure 8: The average number of hops of a successful request

Fig. 8 shows the average number of hops of a successful channel request. The average increases first when the network is lightly-loaded because shorter routes are likely to be used first under our strategy if there are multiple routes which can provide the required end-to-end delay guarantee. As the network load increases, longer routes are found and used by our strategy due to the increased load on the shorter routes. This, in turn, increases the

average number of hops of a channel.

However, as the network load continues to increase, longer routes won't be able to provide the end-to-end delay guarantee, i.e., most channel requests which require a longer route will be rejected. Therefore, we can see the average number of hops decreases rapidly. When the network is heavily-loaded, e.g., 85%, the three lines in Fig. 8 converge because the hop limit becomes unimportant under a such high network load for the same reason mentioned above.

These results demonstrate that our strategy is effective and outperforms the conventional shortest-path strategy. Moreover, with an appropriate choice of hop limit (such as 4 in our simulation), the overhead can also be reduced significantly (35% when the network is lightly-loaded), while providing comparable performance. Fig. 8 has shown that our strategy is able to balance the load of the network by using longer routes when shorter routes are (relatively) congested.

4 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. Our simulation results have also shown this scheme to make significant performance improvements over the conventional shortest-path strategy.

References

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

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

[3] 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.

[4] 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.

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

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

[7] 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.

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

Biographical Sketches

Kang G. Shin is Professor and Director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan. He received his BS degree in electronics engineering from Seoul National University, Korea, and his MS and PhD degrees in electrical engineering from Cornell University, Ithaca, NY, USA.

He has authored/coauthored over 300 technical papers (more than 140 of these in archival journals) and numerous book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing and transportation systems. In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award and in 1989, he also received the Research Excellence Award from The University of Michigan.

He is an IEEE fellow, was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the General Chairman of the 1987 RTSS, the Guest Editor of the 1987 August special issue of *IEEE Transactions on Computers* on Real-Time Systems, a Program Co-Chair for the 1992 *International Conference on Parallel Processing*, and served numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991-93, was a Distinguished Visitor of the Computer Society of the IEEE, an Editor of *IEEE Trans. on Parallel and Distributed Computing*, and an Area Editor of *International Journal of Time-Critical Computing Systems.*

Chih-Che Chou received his Ph.D. degree from the University of Michigan, Ann Arbor, MI in 1994. Currently he is a member of technical staff in AT&T Bell Laboratories. His research interests include real-time operating systems, real-time communication and fault-tolerant distributed systems.