Fault-Tolerant Real-Time Communication in Distributed Computing Systems

Qin Zheng, *Member*, *IEEE Computer Society*, and Kang G. Shin, *Fellow*, *IEEE*

Abstract—The delivery delay in a point-to-point packet switching network is difficult to control due to the contention among randomly-arriving packets at each node and multihops a packet must travel between its source and destination. Despite this difficulty, there are an increasing number of applications that require packets to be delivered reliably within prespecified delay bounds. This paper shows how this can be achieved by using real-time channels which make "soft" reservation of network resources to ensure the timely delivery of real-time packets. We first present theoretical results and detailed procedures for the establishment of real-time channels and then show how the basic real-time channels can be enhanced to be fault-tolerant using the multiple disjoint paths between a pair of communicating nodes. The contribution of the former is a tighter schedulability condition which makes more efficient use of network resources than any other existing approaches, and that of the latter is a significant improvement in fault tolerance over the basic real-time channel, which is inherently susceptible to component failures.

Index Terms—Real-time fault-tolerant communications, point-to-point packet switching networks, deadline scheduling, single-failure-immune (SFI) networks.

1 Introduction

Timely delivery of intertask messages—called *real-time communication*—is essential to the completion of real-time tasks before their deadlines. On one hand, distributed systems with point-to-point interconnection networks are natural candidates for real-time communication because parallel processing and communication, as well as fault tolerance, can be achieved using multiple processors and interconnection paths between every pair of nodes. On the other hand, due to the contention among randomly-arriving messages at each node/link and multihops between the source and destination that a message must travel, it is difficult to guarantee the timely delivery of intertask messages. The main goal of this paper is to remove this difficulty and simultaneously realize the potential of distributed systems for high performance and high reliability.

If messages are broken into packets of the same size, one way to achieve real-time communication is to use circuit-switched transmission based on the Synchronous Transmission Mode (STM), i.e., statically assign time slots of transmission links to each circuit over which real-time communication must be achieved. However, there are two problems associated with circuit-switched transmission. First, since time slots are statically assigned to each circuit, there is a waste of bandwidth associated with the time slots during which no packets are transmitted. Second, it is difficult to establish circuits with different delay and/or bandwidth requirements. Hence, although the circuit-switched transmission has been quite successful for digital telephone communications, it is not suitable for distributed real-time computing systems.

The first problem can be solved by using packet-switched transmission in which link transmission times are dynamically assigned to those connections which have packets to transmit. However, due to the dynamic sharing of link bandwidth among multiple connections, it is difficult to guarantee packet transmission delay bounds in a packet-switched network.

The approach of *real-time channel*, first proposed by Ferrari and Verma [1], then refined by Kandlur et al. [3], is an enhancement to the conventional packet-switched network to provide delay bound guarantees to real-time connections. With the real-time channel approach, each packet is assigned a deadline over each link on its route and the transmission of packets over a link is scheduled according to their deadlines. Using a proper deadline assignment policy, the network will first serve those packets of the channels that require tight delivery delays and/or high link bandwidths. This will solve the second problem of the STM.

Liu and Layland [2] proved the deadline scheduling policy to be "optimal" in the sense that if the transmission of packets can be completed before their deadlines using *any* scheduling policy, so can they using the deadline scheduling policy. Thus, the deadline scheduling policy is best suited for a communication subsystem where all time-constrained packets must be delivered before their deadlines. However, as pointed out in [3], the main problem with the deadline scheduling policy is the difficulty in computing guarantees. In other words, there are no efficient ways to solve the following two problems:

- *The schedulability problem*: Given a set of real-time channels passing through a link, *verify* if all packets of these channels can be delivered over the link before their deadlines, and

- *Q. Zheng is with Argon Networks, 25 Porter Road, Littleton, MA 01460. E-mail: zheng@argon.com.*
- *K.G. Shin is with the Department of Electrical Engineering and Computer Science, Real-Time Computing Laboratory, University of Michigan, Ann Arbor, MI 48109-2122. E-mail: kgshin@eecs.umich.edu.*

Manuscript received 29 June 1995; revised 2 Oct. 1996.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 104590.

- *The minimum guaranteed delay problem*: Given a new real-time channel to be established, *calculate* the minimum delivery delay bound that a link can guarantee for this channel without violating the guarantees of existing real-time channels.

The concept of real-time channel cannot be efficiently implemented without solving these two problems.

Ferrari and Verma [1] obtained a solution to the schedulability problem under the assumption that the summation of the maximum packet transmission times over all real-time channels passing through a link is not larger than the minimum packet interarrival times of these channels. This assumption is quite restrictive in practice, as it limits the traffic types to be serviced. Without using this assumption, Kandlur et al. [3] established a *sufficient* condition to check the schedulability of channels. They first derived the schedulability conditions from the priority scheduling policy. Since any set of channels which are schedulable under the priority scheduling policy are also schedulable under the deadline scheduling policy, this condition is a sufficient schedulability condition for the deadline-driven policy.

It can be proven that, under the assumption of [1], the sufficient condition in [3] is equivalent to that in [1]. So, the former subsumes the latter; that is, [3] can deal with situations where the assumption of [1] fails to hold. However, using sufficient schedulability conditions for establishing real-time channels may still under utilize the network's transmission capacity since violation of the sufficient conditions does not necessarily mean that the channels cannot be established.

In this paper, we study the real-time channel establishment problem and present a schedulability condition which is *both* necessary and sufficient. So, under this condition, the network's transmission capacity can be better utilized in accommodating real-time channels. Based on this condition, an efficient solution to the minimum guaranteed delay problem is also derived and used in dividing the requested end-to-end packet delivery delay of a real-time channel into delay segments, each of which is assigned as the deadline of a link the channel runs through.

One important, yet under explored, problem is the fault tolerance associated with the concept of real-time channel. For ease in controlling the end-to-end packet delays, the static routing approach is used for real-time channels. All packets of a real-time channel are transmitted along the same path. This, unfortunately, is more susceptible to component failures than a dynamic routing approach, since a single component failure may disable the whole channel, while packets can be easily routed around the broken components with a dynamic routing approach.

The failure-handling techniques described in [4] for datagram communication are inadequate for real-time communication because real-time messages cannot be detoured around the failed component on the fly. Techniques for real-time communication in multiaccess LANs, such as the one in [5], are not applicable to multihop point-to-point networks, either.

There have also been proposed forward-recovery approaches, as in [6], [7], where multiple copies of a message are sent via disjoint paths to mask component failures. A variation of these approaches coupled with the error-correction coding scheme can be found in [8]. But all of these approaches cannot guarantee bounded recovery delays.

The method proposed in [9] requires all failures to be broadcast to the entire network. When a source node is notified of the failure of its channel, it tries to establish a new channel from scratch. Since no resource is reserved in advance for the fault tolerance purpose, this method has a small overhead in the absence of faults. However, it does not give any guarantee on failure recovery. The channel reestablishment attempt for failure recovery can be rejected, even when there are sufficient resources, as a result of the contention among several simultaneous recovery attempts. Moreover, successive attempts of channel reestablishment may extend the recovery delay significantly.

In this paper, we will show how a real-time channel can be enhanced at a moderate cost to be immune to any single component failure within the channel. The basic idea is to use a constrained dynamic routing approach which strikes a balance between the reliability of real-time channels and the cost of guaranteeing timely delivery of packets. For real-time communication which can tolerate rare, short-period breakdowns, we show how backup channels can be established with the minimum cost which make the quick reestablishment of real-time channels possible. (See [12] for implementation details of the backup channel approach.)

The paper is organized as follows. Section 2 introduces the concept of real-time channel, solutions to both the schedulability and the minimum guaranteed delay problems, and algorithms for the establishment of real-time channels. Section 3 shows how single-failure-immune (SFI) real-time channels can be established. Section 4 discusses the idea of backup channels for the quick recovery from broken channels. The paper concludes with Section 5.

2 Real-Time Channels

A *real-time channel* is defined as a simplex virtual circuit between the source and destination nodes which guarantees the delivery of packets within a user-requested end-to-end delay bound [1], [3]. The choice of virtual circuits over datagram services is based on their relative ease in controlling packet delivery times. Moreover, communication with virtual circuits turns out to be suited very well for most real-time applications.

Like an ordinary virtual circuit in a packet switching network, there are three phases associated with each real-time channel: channel setup, packet transmission, and channel tear-down. Once the channel is set up, all packets of the real-time channel are transmitted over the same path. The difference between a real-time channel and an ordinary virtual circuit is that a real-time channel guarantees the user-requested end-to-end packet delivery delay, while the latter does not provide such guarantees. Guaranteeing the user-requested end-to-end delay makes the channel setup procedure more complex than that for a virtual circuit. Also, a more complex scheduling policy than the conventional FIFO must be implemented for packet transmissions.

To set up a real-time channel, the requesting process must specify its traffic generation pattern. Because of the

limited capacity of transmission links, no bounded message delivery delay can be guaranteed without this information. We use two parameters, T and C, to describe a traffic pattern, where T is the minimum packet inter-arrival time and C is the maximum packet-transmission time over a link.[1] It is reasonable to assume prior knowledge of these parameters for many applications, such as interactive voice/video transmission and real-time control/monitoring. In other applications where the traffic pattern is less predictable, the estimated values of T and C could be used. A process may exceed its prespecified maximum packet generation rate at the risk that these packets may be delivered with delays longer than the prespecified bound, or may even be discarded, but this particular process will not affect the guarantees of the other existing channels.

Upon receiving the request for establishing a new real-time channel, the service provider must

1) select a route for the channel, and
2) check if the requested end-to-end delay can be satisfied along this route.

The real-time channel establishment request is accepted if this test is passed, and rejected otherwise. We will focus on the second step in the remainder of this section. (The route selection step will be discussed in the next section.)

The end-to-end packet delivery delay is the summation of delays over links and nodes that the channel runs through, which are composed of:

- *Switching delay*: The time needed to move a packet from an input link to an appropriate output link buffer.
- *Scheduling delay*: The time needed to choose a packet in the wait queue for transmission, or insert the packet in an ordered queue, according to a specific scheduling policy.
- *Queuing delay*: The waiting and transmission time of a packet at the transmitter of an output link.
- *Propagation delay*: The time needed for the packet to reach the next node.

The switching delay at a node depends on the switch architecture and the switching technique used by the node. According to the discussion in [10], the data-transfer speed inside a switching node is usually much faster than that of an output link and, thus, the switching delay is negligible as compared to the other delay elements.

The scheduling delay depends on the scheduling policy used. The usual First-In-First-Out (FIFO) policy incurs practically no scheduling delay, as the first packet in the wait queue is always chosen for transmission and a newly-arrived packet is inserted at the end of the queue. However, the overhead incurred by the deadline scheduling policy—the policy to be used in real-time channels—could be significant since the entire queue may have to be searched for packet selection/insertion. Designing a fast packet switch that implements the deadline scheduling policy is an interesting problem of its own right. (See [11] for an example hardware design for deadline scheduling.) In this paper, we assume the availability of such a fast switch and thus assume the scheduling delay to be negligible. The propagation delay depends on the total link length between the source and destination nodes and is constant for a given route, so it can be presubtracted from the requested end-to-end delivery delay.

Because of the conflicts at the output link, queuing delays are inevitable. In the rest of this section, we will discuss how they can be controlled within a certain bound for real-time channels.

Suppose a real-time channel τ_i requires an end-to-end packet delivery delay D_i, and the (physical) route of the channel consists of k links, $\ell_1, \cdots, \ell_k$. One way to meet the end-to-end packet delivery deadline is to guarantee the packet delivery deadline over each link on the route. In other words, if the packet delay over link ℓ_j is guaranteed to be no greater than d_i^j *and* $\sum_{j=1}^{k} d_i^j \le D_i$, then the end-to-end packet delivery delay cannot be greater than D_i. Guaranteeing the end-to-end packet delivery delay in this way, albeit conservatively, makes the problem much easier to solve, because we only need to calculate the delay bound over a single link, rather than over the entire channel.

By amortizing the end-to-end requested delivery delay over the links of its route, a real-time channel τ_i running through a link ℓ_j can be described as a three-tuple $\left(T_i, C_i^j, d_i^j\right)$, where T_i is the minimum packet interarrival time at the link, C_i^j is the maximum packet-transmission time over the link, and d_i^j is the packet delivery delay bound assigned to the link. Without loss of generality, one can assume T_i, C_i^j, and d_i^j to be positive. We will also omit the superscript j when the link number j is immaterial.

A set of channels $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, ..., n$, is said to be *schedulable* over a link if for all $1 \le i \le n$, the maximum queuing delay experienced by channel i's packets over the link is not greater than the requested delay bound d_i.

A real-time channel τ_i can be established in two different ways. The first way is to divide the end-to-end delay D_i into link delay bounds d_i^js before establishing the channel. Then, check the schedulability of the channel over each of the links on the channel's route. If all the checks are positive, τ_i can be established successfully. Otherwise, one must reassign link delays and repeat the procedure. The second way is to calculate the minimum delay bound each of the links on the channel's route can guarantee without affecting the schedulability of existing channels. If the sum of these minimum link-delay bounds is not greater than the user-requested end-to-end delay, the real-time channel can be established. Otherwise, it is impossible to establish the channel at this time unless one chooses an alternative route or the client increases his requested end-to-end delay.

The above two ways of establishing a real-time channel introduce the following two problems.

1. C varies over links with different transmission rates and lengths. Actually, a user gives the maximum packet size and each node determines the value of C for each of its output links.

- *Schedulability Problem*: Given a set of n channels $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, \ldots, n$, running through a link, are they all schedulable?
- *Minimum Guaranteed Delay Problem*: Suppose $n - 1$ channels, $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, \ldots, n-1$, are schedulable over a link. Given a new channel τ_n with the minimum packet interarrival time T_n and the maximum packet-transmission time C_n, what is the minimum value of d_n such that all $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, \ldots, n$, are still schedulable?

Solutions to these problems are presented in the following two theorems.

THEOREM 1. *A set of channels $\tau_i = (T_i, C_i, d_i)$, $i = 1, 2, ..., n$, are schedulable over a link under the preemptive deadline scheduling policy if and only if both of the following hold*:

1) $\sum_{i=1}^{n} C_i/T_i \le 1$.

2) $\forall t \in S$, $\sum_{i=1}^{n} \left\lceil (t - d_i)/T_i \right\rceil^{+} C_i \le t$, *where* $S = \bigcup_{i=1}^{n} S_i$,

$$S_i = \left\{ d_i + nT_i : n = 0, 1, \ldots, \left\lfloor (t_{max} - d_i)/T_i \right\rfloor \right\},$$

$$t_{max} = \max\left\{ d_1, \ldots, d_n, \left\lfloor \sum_{i=1}^{n} (1 - d_i/T_i) C_i \right\rfloor \Big/ \left(1 - \sum_{i=1}^{n} C_i/T_i\right) \right\},$$

and $\lceil x \rceil^{+} = n$ *if* $n - 1 \le x < n$, $n = 1, 2, ...,$ *and* $\lceil x \rceil^{+} = 0$ *for* $x < 0$.

THEOREM 2. *Let* $f(t, d_n) = \sum_{i=1}^{n} \left\lceil (t - d_i)/T_i \right\rceil^{+} C_i$ *and S be the set defined in Theorem 1 with $d_n = C_n$. Then, $d_n = C_n$ is the solution to the minimum guaranteed delay problem if $\forall t \in S$, $f(t, C_n) \le t$. Otherwise, the solution is $d_n = \max\{d^t : t \in G\}$, where $G = S \cap \{t : f(t, C_n) > t\}$ and d^t is computed as* $d^t = C_n + k_f^t T_n + \epsilon_f^t + \epsilon_t^t$, *with* $k_f^t = \left\lceil f(t, C_n) - t/C_n \right\rceil - 1$,

$$\epsilon_f^t = f(t, C_n) - t - k_f^t C_n,\ \epsilon_t^t = t - C_n - k_t^t T_n,\ k_t^t = \left\lfloor (t - C_n)/T_n \right\rfloor.$$

Proofs of these theorems and more results about the schedulability and minimum guaranteed delay problems can be found in [13].

Using Theorems 1 and 2, we have developed the following two algorithms to establish a real-time channel τ_i over a given route consisting of k links, $\ell_1, \cdots, \ell_k$.

Algorithm 1:

STEP 1. Divide the requested end-to-end delay D_i among the links, i.e., assign a link delay bound d_i^j over ℓ_j, $j = 1, \cdots, k$, such that $\sum_{j=1}^{k} d_i^j = D_i$.

STEP 2. Using Theorem 1, check the schedulability of the channel over each link. If all the checks are positive, then the requested real-time channel can be established with the assigned link delays d_i^j s. Otherwise, the channel establishment request is rejected.

Algorithm 2:

STEP 1. Using Theorem 2, calculate the minimum packet delay bounds $d_{min,i}^j$ over link ℓ_j, $j = 1, \cdots, k$.

STEP 2. If $\sum_{j=0}^{k} d_{min,i}^j \le D_i$, then the requested real-time channel can be established. Assign the link delay over ℓ_j to be $d_i^j = d_{min,i}^j + \delta_i/k$, where $\delta_i = D_i - \sum_{j=1}^{k} d_{min,i}^j$. Otherwise, the channel establishment request is rejected.

Algorithm 2 is, in general, superior to Algorithm 1 since it "optimally" divides the end-to-end delay D_i into link delays d_i^j s. In other words, if a real-time channel cannot be established using Algorithm 2, then it cannot be established using Algorithm 1 either under any link-delay assignment policy. However, Algorithm 1 is useful in situations where the network is lightly loaded (thus, the requested channel may always be established) and the client has specific preference on either of the link-delay assignment policies.

We make several remarks on the above results.

1) The description of a channel τ_i over a link with a three-tuple (T_i, C_i, d_i) means that the packet interarrival times at the link are not smaller than a constant T_i. This is true at the first link of the channel if the source node abides by the traffic generation constraints. The packet interarrival times at intermediate links, however, may be smaller than T_i because of the different delays experienced by the packets at the previous links. For this reason, one should use a *logical* packet arrival time t_ℓ—which is defined as the time the packet would have arrived at the link if it had experienced the largest delays (i.e., the requested link delay bounds d_i^j s) at all previous links—to compute the packet's deadline, $t_\ell + d_i$. Clearly, using its logical packet arrival time will not exceed the user-requested packet's end-to-end delay. Also, under heavy traffic conditions, using the logical arrival time will lower the priority of a packet, which has gained time over the previous links (thus, arrived earlier) and allow other tighter-deadline packets (or even non-real-time packets) to be transmitted first. Under light traffic conditions, using the logical arrival time will not delay the transmission of a packet because the packet can be transmitted before its logical arrival time.
2) At the source node, one can use the logical generation time to deal with the case when the source process violates its prespecified traffic constraints. Specifically, if a packet has a length longer than the prespecified value of the corresponding channel, it is broken into one or more packets. If the packet generation rate is greater than the prespecified value, the packets which are generated earlier than specified are assigned deadlines according to the times when they should have been generated. In this way, only those packets that violate the traffic constraints may suffer larger delays, but this violation will not affect the delivery guarantees of other channels.

   This property also enables real-time channels to possess the capability of distributed flow control. Each real-time channel is guaranteed to have a certain network bandwidth; no channel can deprive other channels' guarantees.

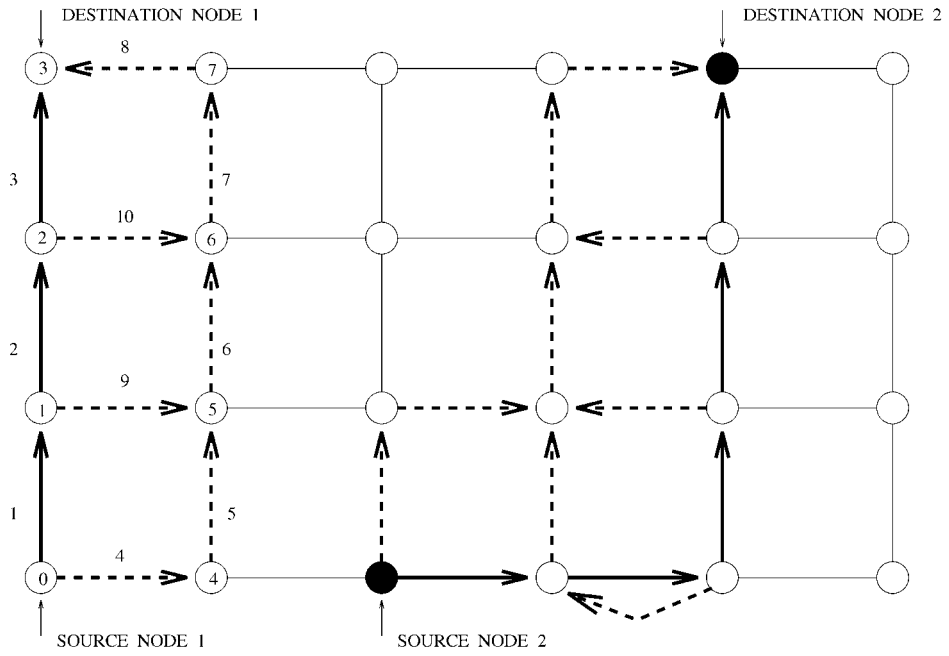


Fig. 1. Two optimal SFI circuits in a mesh network. Solid arrows represent the basic circuits and dashed arrows represent detours.

3) Non-real-time and datagram services can be easily supported with the concept of real-time channel. Over each link, one can set up a special real-time channel τ_0 with large values of T_0 and d_0. All non-real-time and datagram packets are transmitted through channel τ_0. Since T_0 and C_0 are large, τ_0 has little influence on the other real-time channels. Also, non-real-time or datagram packets will be transmitted immediately after transmitting all or most of the real-time packets. So, the network bandwidth is efficiently used. Dynamic routing of datagram packets is also possible by running τ_0 through *every* link of the network. One can then send a datagram packet to an output link over which the packet will be assigned the earliest deadline.

3 Single-Failure-Immune (SFI) Real-Time Channels

In addition to their potential for high performance, high reliability is another attractive feature of point-to-point connected distributed systems. High reliability comes from the multiple processors and interconnection paths existing in a distributed system that provide natural spatial redundancy. However, the real-time channels discussed in the last section do not take advantage of this feature. All packets of a real-time channel are transmitted along the same path, so a single component failure can disable the entire channel.

A natural way to increase the reliability of a real-time channel is to expand the channel with some extra links and nodes, such that packets can be rerouted around faulty components on the original channel. One extreme is to use all links and nodes in the network such that packets can be successfully routed in a timely manner as long as the network remains connected. However, this method is usually very expensive. The cost comes not only from the large number of links/nodes involved, but also from the delay bound requirements over the links. For a real-time channel which remains operational as long as the network is connected, the number of hops a packet may traverse, in the worst case, is very large. The large number of hops from the source to destination node requires each link on the route to provide a very small delay bound so that the end-to-end delay bound can be guaranteed. From Theorem 1, it is easy to see that establishing a real-time channel with a very small requested delay bound over a link is very costly and very likely to be rejected.

By making a trade-off between reliability and cost, we want to establish each real-time channel which is single-failure-immune (SFI). A real-time channel is said to be SFI if it guarantees the timely delivery of a packet as long as the packet encounters no more than one link/node failure on its way to the destination node.

For convenience of presentation, we introduce some definitions first. A communication network is modeled as a directed graph $N = \{V, E\}$, where V is a set of nodes and E is a set of directed links. A *basic circuit* from node v_0 to node v_k in the network is defined as a sequence $C_b = v_0 e_1 v_1 e_2 \ldots e_k v_k$, where v_is are nodes and $e_i = \overrightarrow{v_{i-1} v_i}$ is a directed link from v_{i-1} to v_i. AN SFI *circuit* C_s from node v_0 to node v_k in the network is defined as a basic circuit C_b from v_0 to v_k, augmented with some extra nodes and links, which are called the *detours* of the basic circuit, such that there exists a basic circuit from v_0 to v_k in C_s if no more than one node/link (except the source and destination nodes v_0 and v_k) is removed from C_b. Notice that the removal of a node means the removal of all links incident to/from it. In Fig. 1, links 1-3 and the nodes they connected compose a basic circuit from SOURCE NODE 1 to DESTINATION NODE 1. The basic circuit can be augmented into an SFI circuit with the addition of links 4-10 and the nodes they connected.

To establish an SFI real-time channel, the first step is to find an SFI circuit on which the real-time channel is to be established. A straightforward way to find an SFI circuit from v_0 to v_k is given as follows: First, establish a basic circuit from v_0 to v_k, then break the links and nodes down on the basic circuit one at a time and establish basic circuits from v_0 to v_k in the remaining network. The union of all the basic circuits forms an SFI circuit from v_0 to v_k. Failure of this algorithm means that no such SFI circuits exist.

If an SFI circuit is to be used for the establishment of an SFI real-time channel, some extra features are desirable. From Theorem 2, we see that the existing real-time channels over a link affect the link's ability to guarantee delay bounds for future channels. In establishing a new channel, it is thus desirable to minimize this negative influence on future channels to be established.

There are two ways to measure this influence of an ordinary real-time channel. First, the more links a real-time channel traverses, the more pronounced the influence will become. This is because a larger number of links are involved in establishing a long channel than a short channel. Second, over a single link, the smaller the requested packet delivery delay d_i^j, the more pronounced the influence will become (see Theorem 1). Thus, to reduce a real-time channel's influence on the network's ability to establish future channels, one should run it through as few links as possible and make the requested packet delivery delay over each link as large as possible. Note that the second objective is consistent with the first, because the more links a real-time channel traverses, the smaller the requested packet delivery delay per link would become. Hence, minimum-hop routes are best suited for real-time channels.

Another advantage of minimum-hop routing for real-time channels is the reduction of real-time packets' influence over non-real-time packets. If each real-time packet traverses through a minimum number of links, the total real-time traffic in the network would be minimized. Since transmission priority is usually given to real-time packets, minimizing real-time traffic effectively minimizes its influence on non-real-time packets in the network.

For SFI real-time channels, the influence of the existing real-time channels on the future channel establishments also includes that over the detours. Thus, one should establish an SFI real-time channel over an SFI circuit which needs a minimum number of extra links.

In summary, we have the following two goals in selecting SFI circuits for the establishment of SFI real-time channels:

G1: The basic circuit is a minimum-hop route from the source to the destination node. In the case of link / node break-down, the detours should also be the minimum-hop routes in the remaining network.

G2: Under the constraint of G1, the total number of links on the detours of an SFI circuit should be as small as possible.

An SFI circuit is said to be *optimal* if it achieves the above two goals. It is not difficult to find an optimal SFI circuit in some widely-used regular networks like meshes and hypercubes. Fig. 1 gives an example of two optimal SFI circuits in a mesh network.

For an arbitrary-topology network, however, optimal SFI circuits are not always readily obtainable. The difficulty comes from the existence of multiple minimum-hop basic circuits between two nodes in a network. Different choices of the basic circuit and detours could result in different numbers of extra links needed for an SFI circuit. We propose a heuristic algorithm for finding an SFI circuit as follows:

Algorithm 3:

STEP 1. Set up a minimum-hop basic circuit $C_b = v_0 e_1 v_1 \cdots e_k v_k$ from the source to destination nodes.

STEP 2. Let the set of extra nodes and links $C = \emptyset$. For $i = 1, \ldots, k$, do the following:

STEP 2.1. Remove node v_i from the original network if $i < k$, and link e_k if $i = k$.

STEP 2.2. Establish a minimum-hop basic circuit C_i from v_0 to v_k in the remaining network. At any node, if there are two directions both leading to a minimum-hop circuit, the one to a node which is closer to $C_b \cup C$ is selected. Break a tie using the following rules:

1) choose the one which does not introduce a new link,
2) choose the one which is closer to C_b, and
3) break the tie arbitrarily.

If a basic circuit from v_0 to v_k does not exist, go to Step 3.

STEP 2.3. Suppose C_i intersects a node v_j in C. Then, for $n = i, i + 1$, if

1) there is a basic circuit C_{jn} from v_j to v_n in C and
2) the number of hops from v_j to v_n in C plus the number of hops from v_n to v_k in C_b is not smaller than the number of hops from v_j to v_k in C_i, then remove nodes and links of C_{jn} from C. Add nodes and links of C_i into C.

STEP 3. If the algorithm fails at Step 2, there does not exist an SFI circuit from v_0 to v_k. Otherwise, remove nodes and links of C_b from C. $C \cup C_b$ is an SFI circuit connecting v_0 and v_i with C_b as its basic circuit and C as the set of extra links and nodes.

As an example, we show how the SFI circuit from SOURCE NODE 1 to DESTINATION NODE 1 in Fig. 1 can be established using Algorithm 3. The relevant nodes and links and their labels are shown in Fig. 2.

The sequences of the SFI circuit establishment steps are shown below.

STEP 1. Set up a minimum-hop basic circuit $C_b = v_0 e_1 v_1 e_2 v_2 e_3 v_3$.

STEP 2. The set of extra nodes and links $C = \emptyset$. $i = 1$.

STEP 2.1. Remove node v_1 from the original network.

STEP 2.2. Establish a minimum-hop basic circuit from v_0 to v_3 in the remaining network: $C_1 = v_0 e_4 v_4 e_5 v_5 e_6 v_6 e_{11} v_2\ e_3 v_3$. At node v_6, both e_{11} and e_7 lead to a minimum-hop circuit, but e_{11} is chosen since it leads to a node closer to C_b.

STEP 2.3. $C = C \cup \{\text{links and nodes of } C_1\} = \{e_4, e_5, e_6, e_{11}, e_3, v_0, v_4, v_5, v_6, v_2, v_3\}$.

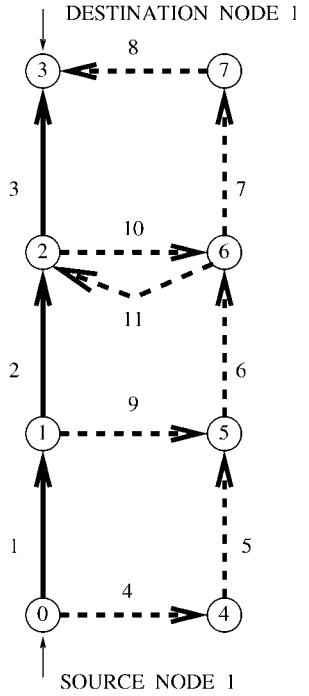


Fig. 2. An example of using Algorithm 3.

STEP 2. $i = 2$.

STEP 2.1. Remove node v_2 from the original network.

STEP 2.2. Establish a minimum-hop basic circuit from v_0 to v_3 in the remaining network: $C_2 = v_0e_1v_1e_9v_5e_6v_6e_7v_7e_8v_3$.

STEP 2.3. $C = C \cup \{\text{links and nodes of } C_2\} = \{e_4, e_5, e_6, e_{11}, e_3, e_1, e_9, e_7, e_8, v_0, v_4, v_5, v_6, v_2, v_3, v_1, v_7\}$.

STEP 2. $i = 3$.

STEP 2.1. Remove link e_3 from the original network.

STEP 2.2. Establish a minimum-hop basic circuit from v_0 to v_3 in the remaining network: $C_3 = v_0e_1v_1e_2v_2e_{10}v_6e_7v_7e_8v_3$.

STEP 2.3. C_3 intersects a node v_6 in C and

1) there is a basic circuit from v_6 to v_2: $C_{62} = v_6e_{11}v_2$, and
2) the number of hops from v_6 to v_2 in C plus the number of hops from v_2 to v_3 equals the number of hops from v_6 to v_3 in C_3 (They both equal 2).

So, remove links and nodes of C_{62} from C. After adding the nodes and links of C_3 to C, we get $C = \{e_4, e_5, e_6, e_1, e_9, e_7, e_8, e_{10}, v_0, v_4, v_5, v_6, v_3, v_1, v_7\}$.

STEP 3. Remove nodes and links of C_b from C, we have $C = \{e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, v_4, v_5, v_6, v_7\}$. Then, C_b is an SFI circuit from v_0 to v_3 with C_b as its basic circuit and C as the set of extra links and nodes.

The heuristic used in the algorithm is that a detour route C_i should be as close to the existing routes $C_b \cup C$ as possible. In this way, C_i will most likely intersect $C_b \cup C$, thus reducing the number of links needed. The purpose of Step 2.3 is to remove any redundant links and nodes. Actually, we can prove Step 2.3 is optimal in the sense that the SFI circuit obtained from Algorithm 3 is irreducible, i.e., it does not contain any redundant links. This result is stated in the following theorem.

THEOREM 3. *Let $C \cup C_b$ be the SFI circuit obtained from Algorithm 3. Then, the removal of any link from C violates G1.*

PROOF. We prove this theorem by contradiction. Suppose a link $e_l = \overrightarrow{v_{l_1} v_{l_2}}$ is removed from C and the remaining network $(C/e_l) \cup C_b$ still satisfies G1.[2]

Suppose e_l is added to C at the ith iteration of Step 2. In other words, link e_l is on the minimum-hop detour C_i from v_i to v_k when link e_{i+1} is broken. Since $(C/e_l) \cup C_b$ still satisfies G1, there is another minimum-hop detour C_i' from v_i to v_k. Let $e_m = \overrightarrow{v_{m_0} v_{m_1}}$ be the first link of C_i' which is not on C_i. Suppose C_i takes a link $e_h \neq e_m$ at node v_{m_0}. Clearly, e_m cannot be in C_b. Otherwise, from Step 2.2, C_i would have taken e_m instead of e_h. Then, there is a $j \neq i$ such that e_m is added to C at the jth iteration of Step 2. In other words, e_m is on the minimum-hop detour C_j from v_j to v_k when link e_{j+1} is broken.

Suppose $j > i$. We claim that C_i must intersect C_b at v_n, $n = j$ or $j + 1$. Otherwise, from Step 2.2 of the Algorithm 3, C_j would have taken e_h instead of e_m. Since both C_i and C_j are minimum-hop detours, the number of hops from v_{m_0} to v_n on C_i plus the number of hops from v_n to v_k on C_b equals the number of hops from v_{m_0} to v_k on C_j. Thus, e_h should have been removed from C at the jth iteration of Step 2.3. This is a contradiction.

Similarly, if $j < i$, it can be proven that e_m should have been removed from C at the ith iteration of Step 2.3. Again, this is a contradiction. □

From Theorem 3, we see that Algorithm 3 guarantees a local optimal solution in the sense that the SFI circuit thus obtained cannot be improved without adding some links / nodes to it.

We now develop two algorithms similar to Algorithms 1 and 2 to establish an SFI real-time channel on an SFI circuit. Let $C_s = C \cup C_b$ be an SFI circuit on which an SFI real-time channel is to be established, and let $C_b = v_0e_1v_1 \cdots e_kv_k$ be the basic circuit from the source node v_0 to the destination node v_k. Suppose there are m links in C_s. Label the links in C as $e_{k+1}, \cdots, e_m$. Let $C_0 = C_b$ and, for $i = 1, 2, \cdots, k$, let C_i be the minimum-hop circuit from v_0 to v_k in C_s when link e_i is broken. Define a $(k + 1) \times m$ circuit-link matrix $M = (m_{ij})_{k \times m}$ as follows:

$$m_{ij} = \begin{cases} 1 & \text{if } C_i \text{ contains } e_{j+1} \quad i = 0, \ldots, k. \\ 0 & \text{otherwise} \qquad\qquad\quad j = 0, \ldots, m-1. \end{cases}$$

2. The notation S_1/S_2 denotes the set containing all the elements of S_1 which are not in S_2.

Suppose link e_i guarantees a delay bound d_i. Then, an SFI real-time channel can be established with an end-to-end delay bound D if and only if the following delay inequality is satisfied:

$$M\begin{pmatrix} d_1 \\ \vdots \\ d_m \end{pmatrix} \le \begin{pmatrix} D \\ \vdots \\ D \end{pmatrix}. \quad (1)$$

Then, similar to Algorithms 1 and 2, we have the following two ways to establish an SFI real-time channel over an SFI circuit.

Algorithm 4:

STEP 1. For $i = 1, 2, \cdots, m$, assign a link delay bound d_i over e_i, such that (1) is satisfied.

STEP 2. Using Theorem 1, check the schedulability of the channel over each link. If all the checks are positive, the requested real-time channel can be established with the assigned link delays d_is. Otherwise, the channel establishment request is rejected.

Algorithm 5:

STEP 1. Using Theorem 2, calculate the minimum packet delay bound $d_{min,i}$ over link e_i, $i = 1, \cdots, k$.

STEP 2. Let $d_i = d_{min,i}$, $i = 1, \cdots, m$. If (1) is satisfied, the requested real-time channel can be established. Assign the link delay over e_j to be $d_i = d_{min,i} + \delta_i$, where δ_is satisfy

$$M\begin{pmatrix} \delta_1 \\ \vdots \\ \delta_m \end{pmatrix} \le \begin{pmatrix} D \\ \vdots \\ D \end{pmatrix} - M\begin{pmatrix} d_{\min,1} \\ \vdots \\ d_{\min,m} \end{pmatrix}. \quad (2)$$

Otherwise, the channel establishment request is rejected.

Since (2) cannot determine a unique solution, we need to give a rule to choose one of the solutions. The reason to increase the delay bound over link e_i from $d_{min,i}$ to $d_{min,i} + \delta_i$ in Algorithm 5 is to reduce the channel's influence on the link's ability to establish more real-time channels. The value of δ_i represents the degree of influence reduced. Thus, with respect to a single link e_i, one should set δ_i as large as possible. However, since the maximization of δ_i must be done under the constraint of (2), an increase of δ_i may cause the decrease of δ_j of another link e_j. To make the whole network evenly loaded, we use the following max-min rule to choose a solution from (2).

Max-min Rule: Among all solutions satisfying (2), choose the one whose smallest element, i.e., $\min_{1 \le i \le m} \delta_i$, has the maximum value. If there is more than one solution satisfying this rule, choose the one whose second smallest element has the maximum value. Repeat this process until a unique solution is obtained.

The max-min rule can be easily implemented with the following algorithm.

Algorithm 6:

STEP 1. Initialize the set of variables to be determined as $S = \{\delta_1, \cdots, \delta_m\}$.

STEP 2. For all $\delta_i \in S$, replace δ_i with a single variable δ in (2). Calculate the maximum value of δ satisfying (2). Notice that (2) contains m inequalities, all elements of M are either 0 or 1, and the maximum value of δ makes at least one inequality become an equality. Remove all δ_is from S which are contained in the equality and set them to be the obtained maximum value of δ.

STEP 3. If $S = \emptyset$, stop. Otherwise, go to Step 2.

As an example of using Algorithms 5 and 6, we show how an SFI real-time channel can be established over the SFI circuit from source node 1 to destination node 1 in Fig. 1. Suppose this real-time channel is to be established in an otherwise idle network and has the minimum packet interarrival time $T = 100$, maximum packet transmission time $C = 5$, and requested end-to-end packet delivery delay $D = 60$. The labels of the links on the SFI circuit are shown in Fig. 1. Then, the circuit-link matrix is

$$M = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}. \quad (3)$$

From Theorem 2, we have $d_{min,i} = 5$ for $1 \le i \le 10$. Thus, the right-hand side of (2) equals $(45\ 35\ 35\ 35)^T$. Using Algorithm 6, we first set $S := \{\delta_1, \cdots, \delta_{10}\}$. Replacing all elements in S with a single variable δ, (2) becomes

$$\begin{pmatrix} 3 \\ 5 \\ 5 \\ 5 \end{pmatrix} \delta \le \begin{pmatrix} 45 \\ 35 \\ 35 \\ 35 \end{pmatrix}. \quad (4)$$

The maximum value δ is then 7, with which the second, third, and fourth inequalities of (2) become equalities which contain all δ_is but δ_3. Thus, at the next iteration of Step 2 in Algorithm 6, $S = \{\delta_3\}$. Replacing δ_3 with δ and setting all other variables in (2) to be 7, it becomes $14 + \delta \le 45$. The maximum value of δ is 31.

Thus, the solution from Algorithm 6 is $(\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_7, \delta_8, \delta_9, \delta_{10}) = (7, 7, 31, 7, 7, 7, 7, 7, 7, 7)$. Using Algorithm 5, the SFI real-time channel is established with link delay bounds $(d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10}) = (12, 12, 36, 12, 12, 12, 12, 12, 12, 12)$.

4 BACKUP CHANNELS FOR REAL-TIME CHANNELS

In the last section, we discussed how real-time channels can be enhanced to be immune to single point failures. For real-time communication that can tolerate rare short-period breakdowns, a less expensive way to increase the reliability of a real-time channel is to set up backup channels. This method is also applicable to cases where no SFI circuits exist from the source to destination nodes due to the poor connectivity of the network. The idea of backup channels works as follows:

- Each real-time channel is composed of a *primary* channel and a number of *backup* channels. Under the normal circumstance, the primary channel is used for packet transmission while keeping the backup channels idle or unused. In case the primary channel gets disabled by a component failure, one of the backup channels is promoted to the primary channel (thus taking over the transmission task).

- Both the primary and backup channels are established simultaneously using the procedures described in Section 2. In case a primary or backup channel cannot be established, the system is allowed to *tear down* some of the existing backup channels based on a scheme that will be described later.

Using backup channels, the long channel reestablishment overhead can be avoided when the primary channel is disabled. The cost of backup channels is minimal because:

- Backup channels are not used to transmit redundant packets under the normal circumstance, thus unaffecting the packets in other channels, as well as non-real-time traffic, and
- The number of real-time channels that a network can accommodate is not reduced since the backup channels can be removed whenever there is a shortage of network resources in establishing a new real-time channel.

As compared to the case of replicating real-time channels, there are two problems in implementing the idea of backup channels. First, there is a delay in switching to a backup channel when a fault occurs to the primary channel. The main source of this delay is associated with fault detection and channel switching. The real-time packets transmitted during this period could be lost. Second, there is no guarantee on the number of backup channels that each real-time channel can have. In the rest of this section, we will address how these two problems can be alleviated.

The fault detection time can be reduced by using an "acknowledgment channel" for each real-time channel. A channel fault can be detected quickly if the source node does not receive an ack in a certain period after transmitting a packet. Note that an acknowledgment channel usually costs far less than a real-time channel since it deals with a shorter packet size, a larger requested delivery delay (depending on the required fault detection time), and/or a longer packet interarrival time (several packets can be acknowledged at a time).

The channel switch time is the time needed to find a nonfaulty backup channel and promote it to the primary channel. If there is at least one nonfaulty backup channel, this process is quite fast. One can send multiple copies of a packet through all the backup channels of the now disabled real-time channel, and choose the one which delivered the packet correctly. If, unfortunately, all the backup channels are faulty, there is no choice but to execute the time-consuming channel establishment procedure.

The second problem of using backup channels comes from the fact that a backup channel may be removed in order to accommodate future real-time channels. A real-time channel may have many backup channels when the network is "lightly loaded" (in the sense that there do not exist many real-time channels in the network). As more and more real-time channels are added, the number of backup channels may decrease or even become zero. Thus, the use of backup channels provide no guaranteed fault tolerance for the primary channels. An important question is then how to manage the establishment and removal of backup channels such that the primary channels can be backed up as much as possible. To this end, the following three questions must be answered:

Q1: For each real-time channel, how many backup channels should be established?

Q2: How should the channels be routed?

Q3: Which channel removal policy should be used?

A straightforward answer to Q1 is that the more backup channels the better. A real-time channel is more likely to find a nonfaulty backup channel if it is backed up by many channels. However, there is a limitation to the number of backup channels which can be established. Establishing too many backup channels will complicate the channel establishment procedure. Also, the system gains little by establishing channels which have a large number of common links and nodes, because a single component failure would bring all of them down. For this reason, we restrict the primary and backup channels of a real-time channel to run through disjoint paths, and the number of backup channels to be established is thus the maximum number of disjoint paths minus one.

As to the routing problem, we argued in Section 3 that the minimum hop routing is preferable for real-time channels. The primary channel and backup channels can be set up sequentially by establishing one at a time, then removing all the intermediate nodes and links it passes through from consideration for setting up next channels, and establishing the next backup channel in the remaining network, and so on.

The third question consists of two parts:

1) Which channels are allowed to be removed?
2) Which of the removable channels should actually be removed?

As to Part 1, establishment of a new channel should only be allowed to remove only those channels which are less important than itself. There are two ways to compare the relative importance of backup channels:

1) The backup channels of a critical real-time channel are more important than those of a less critical one, and
2) For two real-time channels of the same criticality, the backup channels of the one with more backup channels are less important than those of the other real-time channel.

To this end, one may assign a criticality number C to each real-time channel and then assign a rank $R = f(C, k)$ to its kth backup channel, where $f(C, k)$ is an increasing function of C and a decreasing function of k. One such choice is $f(C, k) = C - k$. So, we have a channel removal strategy which allows for tearing down only lower-rank backup channels. Since the establishment of a primary channel is always allowed to tear down backup channels, the rank of all primary channels is assigned to be ∞.

The answer to the second part of the question is that one should remove as few important backup channels as possible. In other words, if there is a choice between two backup channels, the one with lower rank should be removed. This can be done with the following algorithm.

Algorithm 7: Channel Removal Strategy

STEP 1. Establish the new channel using Algorithm 2 without removing any existing backup channels. Terminate if successful. Otherwise, go to Step 2

STEP 2. Remove all the backup channels on the route having lower ranks than the new one, except those with link delay bounds larger than t_m, the time that achieves the maximum of $d_n = \max\{d^t : t \in G\}$ in Theorem 2 (the removal of these channels will not change the link-delay bound of the new channel).

STEP 3. Establish the new channel using Algorithm 2. If it is still not successful, the new channel cannot be established and those channels removed in Step 2 are restored. Otherwise, go to Step 4.

STEP 4. Starting from the backup channels with the highest rank, reestablish the backup channels removed in Step 2. This step reduces the number of backup channels removed.

We now give an example to show how to establish a fault-tolerant real-time channel.

Consider a ring network with five stations connected by five duplex links (link i connects station i and $(i + 1)$ mod 5). All real-time channels have the same minimum packet interarrival time $T_i = 100$ and maximum packet transmission time $C_i = 5$. Suppose three backup channels τ_{1b}, τ_{2b}, τ_{3b} have been established in the network with the following information on source (*src*) and destination (*dst*) stations, user-requested end-to-end delay D_i, channel rank R_{ib}, and link-delay bound d_i^j. (To show the procedure of Algorithm 7 clearly, we do not consider the existing primary channels, since the primary channels are not allowed to be torn down.)

τ_{1b}: src = 3 dst = 0 $D_1 = 15$ $R_{1b} = 1$ $d_{1b}^3 = 10$ $d_{1b}^4 = 5$

τ_{2b}: src = 2 dst = 0 $D_2 = 10$ $R_{2b} = 2$ $d_{2b}^1 = 5$ d_{2b}^0

τ_{3b}: src = 3 dst = 0 $D_3 = 30$ $R_{3b} = 1$ $d_{3b}^2 = 7$ $d_{3b}^1 = 12$ $d_{3b}^0 = 11$.

Now, we want to establish a new real-time channel τ_4 of criticality 4 with *src* = 3, *dst* = 0, $D_4 = 15$.

The primary channel of τ_4 takes the minimum-hop route $\ell_3 \rightarrow \ell_4$. Using Algorithm 2, one can establish the primary channel as:

$$\tau_{4p}: \text{src} = 3 \text{ dst} = 0 \; D_4 = 15 \; R_{4p} = \infty \; d_{4p}^3 = 5 \; d_{4p}^4 = 10.$$

After removing ℓ_3 and ℓ_4 from the network, the network contains only one more connection between station 3 and station 0. So, τ_4 can have, at most, one backup channel τ_{4b}. Since the criticality of τ_4 is 4, the rank of its first backup channel is $R_{4b} = 4 - 1 = 3$. Setting up τ_{4b} on the route $\ell_2 \rightarrow \ell 1 \rightarrow l0$ with Algorithm 2 results in rejecting the channel request. Hence, all other backup channels on the route with lower ranks lower than τ_4, i.e., τ_{2b} and τ_{3b}, are removed. After this removal, τ_{4b} can be successfully established as:

τ_{4b}: src = 3 dst = 0 $D_4 = 15$ $R_{4b} = 3$ $d_{4b}^2 = 5$ $d_{4b}^1 = 5$ $d_{4b}^0 = 5$.

The next step is to reestablish those backup channels removed, starting from the one with the highest rank. Using Algorithm 2, the establishment of τ_{2b} is rejected. So, τ_{2b} must be removed and τ_{3b} is reestablished as:

τ_{3b}: src = 3 dst = 0 $D_3 = 30$ $R_{3b} = 1$ $d_{3b}^2 = 10$ $d_{3b}^1 = 10$ $d_{3b}^0 = 10$.

Consequently, after establishing a new real-time channel τ_4, the network has the following four established channels:

τ_{1b}: src = 3 dst = 0 $D_1 = 15$ $R_{1b} = 1$ $d_{1b}^3 = 10$ $d_{1b}^4 = 5$
τ_{3b}: src = 3 dst = 0 $D_3 = 30$ $R_{3b} = 1$ $d_{3b}^2 = 10$ $d_{3b}^1 = 10$ $d_{3b}^0 = 10$
τ_{4b}: src = 3 dst = 0 $D_4 = 15$ $R_{4b} = 3$ $d_{4b}^2 = 5$ $d_{4b}^1 = 5$ $d_{4b}^0 = 5$
τ_{4p}: src = 3 dst = 0 $D_4 = 15$ $R_{4p} = \infty$ $d_{4p}^3 = 5$ $d_{4p}^4 = 10$.

5 CONCLUSION

We have addressed the problem of fault-tolerant real-time communication in distributed point-to-point connected systems. We analyzed the components of end-to-end packet delivery delay and showed how this delay can be controlled to be below a prespecified value by establishing real-time channels. To increase the reliability of real-time channels, we showed how they can be enhanced to be single-failure-immune. One can also establish backup channels with the minimal cost to which a broken real-time channel can quickly switch.

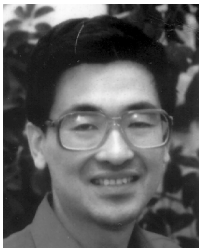
ACKNOWLEDGMENT

The work reported in this paper was supported in part by the U.S. Office of Naval Research under Grant N00014-94-1-0229 and the U.S. National Science Foundation under Grant MIP-9203895. Any opinions, findings, and recommendations expressed in this publication are those of the authors, and do not necessarily reflect the views of the funding agencies. A portion of this paper was presented at the 22nd International Symposium on Fault Tolerant Computing, Boston, Massachusetts, July 8-10, 1992.

REFERENCES

[1] D. Ferrari and D.C. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE J. Selected Areas Comm.*, pp. 368-379, 1990.
[2] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 46-61, 1973.
[3] D.D. Kandlur, K.G. Shin, and D. Ferrari, "Real-Time Communicaiton in Multi-Hop Networks," *Proc. 11th Int'l Conf. Distributed Computing Systems*, pp. 300-307, May 1991. Also appeared in *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 10, pp. 1,044-1,056, Oct. 1994.
[4] D. Comer, *Internetworking with TCP/IP*. Prentice Hall, 1995.
[5] B. Chen, S. Kamat, and W. Zhao, "Fault-Tolerant Real-Time Communication in FDDI-Based Networks," *Proc. Real-Time Systems Symp.*, 1995.
[6] P. Ramanathan and K.G. Shin, "Delivery of Time-Critical Messages Using a Multiple Copy Approach," *ACM Trans. Computer Systems*, vol. 10, no. 2, pp. 144-166, May 1992.
[7] B. Kao, H. Garcia-Molina, and D. Barbara, "Aggressive Transmissions of Short Messages Over Redundant Paths," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 1, pp. 102-109, Jan. 1994.
[8] A. Banerjea, "Simulation Study of the Capacity Effects of Dispersity Routing for Fault Tolerant Realtime Channels," *Proc. ACM SIGCOMM Symp.*, pp. 194-205, Aug. 1996.

[9] A. Banerjea, C.J. Parris, and D. Ferrari, "Recovering Guaranteed Performance Service Connections from Single and Multiple Faults," Technical Report TR-93-066, Computer Science Division, Univ. of California at Berkeley, 1993.
[10] I. Cidon, I. Gopal, G. Grover, and M. Sidi, "Real-Time Packet Switching: A Performance Analysis," *IEEE J. Selected Areas Comm.*, vol. 6, no. 9, pp. 1,576-1,586, 1988.
[11] J. Rexford, J. Hall, and K.G. Shin, "A Router Architecture for Real-Time Point-To-Point Networks," *Proc. 23rd Int'l Symp. Computer Architecture*, pp. 237-246, May 1996.
[12] S. Han and K.G. Shin, "Fast Restoration of Real-Time Communication Service from Component Failures in Multi-Hop Networks," *Proc. ACM SIGCOMM97*, pp. 77-88, Sept. 1997.
[13] Q. Zheng and K.G. Shin, "On the Ability of Establishing Real-Time Channels in Point-To-Point Connected Packet Switching Networks," *IEEE Trans. Comm.*, vol. 42, nos. 2/3/4, pp. 1,096-1,105, Feb./Mar./Apr. 1992.



Qin Zheng (S'89-M'91) received the BS and MS degrees in electrical engineering from the University of Science and Technology of China in 1982 and 1985, respectively, and the PhD degree in electrical engineering and computer science from the University of Michigan in 1993.

He worked with Mitsubishi Electric Research Laboratory from 1993 to 1997, where he conducted research in the area of traffic management for ATM and other high speed comminication networks, and developed a Mitsubishi Electric's second generation ATM network interface chip. Currently, he is with Argon Networks working on a 2.488 Gbps multiservice switching system. Dr. Zheng holds six patents and has published more than twenty technical papers in the area of real-time fault-tolerant communication.

Kang G. Shin received the BS degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and both the MS and PhD degrees in electrical engineering from Cornell University, Ithaca, New York, in 1976 and 1978, respectively. From 1978 to 1982, he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the U.S. Airforce Flight Dynamics Laboratory, AT&T Bell Laboratories, the Computer Science Division within the Department of Electrical Engineering and Computer Science at the University of California at Berkeley, the International Computer Science Institute, Berkeley, California, the IBM T.J. Watson Research Center, and with the Software Engineering Institute at Carnegie Mellon University. He also chaired the Computer Science and Engineering Division in the Electrical Engineering and Computer Science Department at the University of Michigan for three years beginning in January 1991. Currently, he is a professor and director of the Real-Time Computing Laboratory in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, Michigan.

Dr. Shin has authored and coauthored more than 450 technical papers (about 170 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. He has coauthored (jointly with C.M. Krishna) a textbook titled *Real-Time Systems* (McGraw-Hill, 1997). In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award for a paper on robot trajectory planning. In 1989, he received the Research Excellence Award from The University of Michigan. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are investigating various issues related to real-time and fault-tolerant computing.

He has also been applying the basic research results of real-time computing to multimedia systems, intelligent transportation systems, embedded systems, and manufacturing applications ranging from the control of robots and machine tools to the development of open architectures for manufacturing equipment and processes. (The latter is being pursued as a key thrust area of the newly established U.S. National Science Foundation's Engineering Research Center on Reconfigurable Machining Systems.)

Dr. Shin is a fellow of the IEEE. He was the program chairman of the 1986 *IEEE Real-Time Systems Symposium* (RTSS), the general chairman of the 1987 RTSS, a program cochair for the 1992 *International Conference on Parallel Processing*, and has served on numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991-1993 and was a distinguished visitor of the Computer Society of the IEEE. He served as the guest editor of the August 1987 special issue of the *IEEE Transactions on Computers* on real-time systems, was an editor of the *IEEE Transactions on Parallel and Distributed Computing*, and an area editor of the *International Journal of Time-Critical Computing Systems*.