A Simple Distributed Loop-Free Routing Strategy for Computer Communication Networks

Kang G. Shin, *Fellow, IEEE,* and Chih-Che Chou

Abstract—**The loops resulting from either component failures or load changes in a computer communication network degrade the performance and the adaptability of conventional distributed adaptive routing strategies, such as ARPANET's previous routing strategy (APRS). In this paper, we develop a distributed loop-free routing strategy by adding only one additional piece of information—the total number of minimum-delay paths—to the commonly used routing messages and tables. Most conventional approaches to the looping problem suffer high overheads in time and space because each message must either include the first several nodes of its path or trace the entire path to detect a loop. By contrast, the proposed routing strategy requires only easily obtainable information, yet removes loops completely. It is far more efficient in both time and space than its conventional counterparts, especiallyTable Ifor sparse computer networks. The correctness of the proposed strategy is proved, and several illustrative examples are given. The performance of this strategy is shown to be always better than, or at least as good as, that of APRS and any multiorder routing strategies, where the *order* of a routing strategy is determined by the amount of routing information carried in each routing message.**

Index Terms—**ARPANET, computer communication networks, distributed adaptive routing, loop-free or fault-tolerant routing strategy, network delay tables, routing messages.**

I. Introduction

THE prime importance of message routing to the performance of any distributed computing system has led to the development of numerous routing strategies [2]–[4], [7]–[9], [11]–[13]. Adaptive routing is more reliable and efficient than nonadaptive routing because the former can dynamically adjust itself to network changes as a result of component failures and/or load changes. However, in order to make correct routing decisions, each node must have up-to-date information about the network changes. Obviously, it is too costly for each node to maintain and update the information of the entire network, such as the current routing strategy used in ARPANET, especially when the network condition changes often. Although the current APARNET routing strategy can always send messages via optimal (minimum-delay) paths provided each node has complete network information, it requires a large amount of memory to store the information of the entire network and wastes a large portion of network bandwidth to exchange routing messages. Hence, it is desirable for each node to maintain only minimal information which is sufficient to make correct routing decisions.

Many distributed adaptive routing strategies have been reported in the literature, such as the APRANET's previous routing strategy (APRS) [1], [15] which seems to be acceptable for most packet-switched networks due to its simple implementation. However, the problem of looping messages in case of network-delay changes or node/link failures degrades the performance and adaptability of such routing strategies. To remedy this deficiency, we shall, in this paper, develop a distributed adaptive *loop-free* routing strategy which requires as little information as possible.

Several solutions to the looping problem have been proposed, including the TIDAS network [3] and multiorder strategies [12], [13]. The authors of [12], [13] proposed a loop-free algorithm which is somewhat similar to [3] for general networks. In their algorithm, the complete information on the path from the source to the destination is included in the routing messages and tables. Although the looping problem can be resolved completely by this algorithm, the size of the routing message and the memory required to store the routing tables are proportional to the diameter of the network. Hence, it will induce very high operational overheads, especially when a *high-order* strategy [13] is used, where the *order* of a routing strategy is determined by the amount of routing information carried in each routing message. Obviously, there is a tradeoff between the operational overhead and the looping delay.

Another similar approach [4]—which includes only the first node of a shortest path, instead of the entire path, in the routing messages and tables—can solve the looping problem without the same overhead in [12], [13]. However, it increases the time complexity in generating the routing messages because a node needs to search its entire routing table for possible loops. Thus, in the worst case, it must go through each entry in the routing table solely for generating the routing message for each pair of a neighbor node (of the source) and the destination node.

A different approach [9] uses a synchronization phase to solve the looping problem, but it incurs an additional cost of synchronization. Moreover, when no loop is encountered, its performance is worse than APRS. There are some other algorithms propsed in [5], [6], and [14], which also have the same major features as APRS, but which still suffer such inherent drawbacks as poor adaptability and inefficiency.

Manuscript received August 22, 1991; revised June 12, 1992. This work was supported in part by the National Science Foundation under Grant MIP-9012549 and the Office of Naval Research under Contract N00014-92-J-1080. 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.

The authors are with the Real-Time Computing Laboratory, Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 9213608.

Another variation of algorithms which can also reduce the possibility of looping are least-hop routing algorithms [10]. Although they are relatively simpler than the least-delay routing strategy, they can work only under the assumption that the least-hop path is the least-delay path.

Most of the above routing strategies share a common assumption that link delays (including transmission and queueing delays) change relatively slowly compared to the rate of updating routing tables. Our strategy also adopts this assumption. However, this assumption does not limit the ability of our strategy to adapt itself to the dynamic changes of the network as long as the rate of delay change is smaller than the rate of updating routing tables. In order to solve the looping problem completely and avoid the high overheads in time and space, we propose a very simple but effective strategy, called the *order one loop-free algorithm*, which can effectively deal with node or link failures, occasional network structural changes, and link-delay changes.

The paper is organized as follows. In Section II, both APRS and the proposed routing strategy are described, and the correctness of the proposed strategy is proved. The operational overheads of the proposed strategy are analyzed in Section III. The performance analysis of our strategy and its comparison with APRS and multiorder strategies are treated in Section IV. Simulation results are presented in Section V, and the paper concludes with Section VI.

II. Description of the Proposed Routing Strategy

For an n-node computer network, let N_i represent a host computer node, and let $L_{i,j}, 1 \leq i,j \leq n, i \neq j$ be the communication link from N_i to N_j. Also, let SP_{i_0,i_k} be the set of all paths from N_{i_0} to N_{i_k} in the network; then a path $P \in SP_{i_0,i_k}$ is expressed by an ordered sequence of nodes $(N_{i_0}, N_{i_1}, \cdots, N_{i_k}), 1 \leq i_j \leq n, j \in \{0, 1, \cdots, k\}$, and nodes on the path are visited in that order. Let A_i be the set of all nodes adjacent to N_i, that is, there is a link $L_{i,j}$ from N_i to every $N_j \in A_i$.

Because our strategy is similar to APRS except for adding an additional piece of information—the number of minimum-delay paths—to each entry of a routing message, we briefly describe APRS first. Under APRS, the path from one node to every other node is not determined in advance. Instead, every node maintains a *delay table* to record the minimum delay via each of its links to every other destination. The *minimum-delay table* is exchanged periodically (once every 128 ms for APRS) as a routing message between each pair of adjacent nodes, containing the delays of the optimal paths from a node to all the other nodes. Upon receiving a new routing message, each node updates its own routing tables and derives a new minimum-delay table which will be used to route messages and will also be sent to all its neighbors as a routing message for the next (exchange) time interval.

Let $D_{i/j,d}(m)$ denote the delay from N_i via N_j to N_d in the delay table of N_i under APRS during the time interval $[m, m+1)$, and let $DL_{i,j}(m)$ denote the delay of link $L_{i,j}$ at time m, where the time interval between two successive routing-message exchanges is defined as one unit of time. For

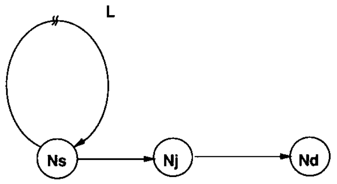


Fig. 1. L is a source loop for the path from N_s going through L, then via N_j to N_d.

simplicity, $DL_{i,j}$ is used to denote the delay of $L_{i,j}$ when time dependency is immaterial. Note that $DL_{i,j}(m)$ includes the transmission delay, propagation delay, and queueing delay. Also, let $OP_{s,d}(m)$ be the minimum-delay (optimal) path from N_s to N_d in the delay table of N_s during the interval $[m, m+1)$, and let $DOP_{s,d}(m)$ be the delay of that path. However, the information kept in the delay table is not always up to date because there might be a component failure which is not immediately known to the source node N_s. That is, $OP_{s,d}(m)$ may *not* be the actual optimal path from N_s to N_d because its routing table could contain obsolete information. According to APRS, the following relationships must hold:

$$D_{i/j,d}(m) = DL_{i,j}(m) + DOP_{j,d}(m-1) \qquad \forall N_j \in A_i \tag{1}$$

$$DOP_{i,d}(m) = \min_{N_j \in A_i} \{D_{i/j,d}(m)\}. \tag{2}$$

However, APRS cannot prevent looping in case of link/node failures and/or load changes. In order to eliminate looping effects, several solutions are proposed by modifying APRS's way of constructing the minimum-delay tables. Unfortunately, most of them result in high operational overheads in memory and/or time. In order to eliminate the looping problem and avoid the excessive operational overheads, we propose a strategy that requires only one additional piece of easily obtainable information—the total number of optimal paths—for each pair of source and destination nodes in a routing message, and some additional simple procedures for constructing the routing messages and updating the delay tables.

Definition 1: A *source loop* of a path is the loop which starts and ends at a node which is the starting node of that "path." For example, L in Fig. 1 is a source loop for the "path" from N_s running through L, then via N_j to N_d, but not a source loop of the path $N_s \rightarrow N_j \rightarrow N_d$.

Under APRS, if there are no paths containing a source loop in the delay tables of all nodes in the network, then all the paths determined by the delay tables do not contain any loop. Therefore, we want to prevent all source loops when constructing routing messages and updating routing tables. Every node sends a routing message to each of its neighbors just as in APRS. Unlike APRS, however, each $N_j \in A_i$ sends N_i the minimum-delay *loop-free* paths as optimal paths for all other nodes in the network which do not pass through N_i.

Since a subpath of an optimal (minimum-delay) path is also an optimal path, the following three cases are sufficient to determine whether $OP_{k,d}$ passes through N_j or not. Let n_{pq} be the number of optimal paths from N_p to N_q with delay d_{pq}.

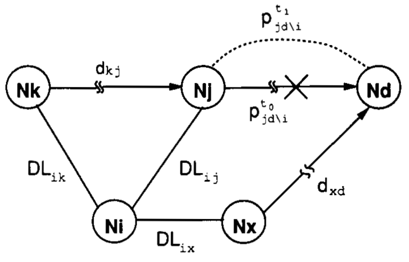


Fig. 2. A known faulty path of N_i.

1) If $d_{kd} \neq d_{kj} + d_{jd}$, then no optimal path from N_k to N_d runs through N_j.

2) If $d_{kd} = d_{kj} + d_{jd}$ and $n_{kd} = n_{kj}n_{jd}$, then all optimal paths from N_k to N_d pass through N_j.

3) If $d_{kd} = d_{kj} + d_{jd}$ and $n_{kd} < n_{kj}n_{jd}$, then some $(n_{kj}n_{jd})$ of the optimal paths from N_k to N_d pass through N_j, but others $(n_{kd} - n_{kj}n_{jd})$ do not.

In order to cope with link/node failures and link-delay changes, we introduce the concept of a *known* faulty path as follows.

For any destination $N_d \neq N_i$, a neighbor node N_k of N_i (i.e., $\in A_i$) is said to have *property A* if there is another neighbor N_j of N_i such that $N_k \to \cdots \to N_j \to \cdots \to N_d$ is a loop-free minimum-delay path from N_k to N_d *without* going through N_i (see Fig. 2). Let S_i represent the set of nodes in A_i which have the property A, and let $\overline{S}_i = A_i \backslash S_i$. At time t, let $SP^t_{jd\backslash i}$ be the set of all optimal paths from $N_j \in A_i$ to N_d without passing through $N_i, n^t_{jd\backslash i} = |SP^t_{jd\backslash i}|$, and let $P^t_{jd\backslash i}$ be a representative path in $SP^t_{jd\backslash i}$.

Suppose at time t_0 there had occurred a link failure in $P^{t_0}_{jd\backslash i}$ which was then detected by $N_j \in A_i$ at time $t_1 > t_0$. At time $t_1 + 1, N_j \in A_i$ will inform N_i of this link failure by sending N_i a new set of optimal paths from N_j to N_d, denoted by $SP^{t_1}_{jd\backslash i}$. The path $P^{t_0}_{jd\backslash i}$ is now a faulty path known to N_i or called an N_i's *known faulty path.*

Likewise, at time $t_2 \geq t_1 + 1, N_k \neq N_j$ is informed of the link failure in $P^{t_0}_{kd\backslash i}$ because in N_k's delay table, $N_k \to \cdots \to N_j \to \cdots \to N_d$ was the optimal path from N_k to N_d at time $t < t_2$. Finally, at time $t_2 + 1, N_k$ informs N_i of the link failure in $P^{t_0}_{kd\backslash i}$ by sending N_i a new set of optimal paths, $SP^{t_2}_{kd\backslash i}$.

During $[t_1 + 1, t_2 + 1)$, the information of the faulty paths in $SP^{t_0}_{jd\backslash i}$ can be used to reject the broken path $P^{t_0}_{kd\backslash i}$ which is specified in the routing message sent from N_k to N_i because at time $t_1 + 1, N_i$ has already been informed of the link failure in $P^{t_0}_{jd\backslash i}$, which is a subpath of $P^{t_0}_{kd\backslash i}$. In the proposed algorithm, the faulty paths information will be saved in N_i's routing table to avoid loops.

N_i's known faulty paths are defined formally as follows. Assume that N_d is an arbitrary destination node in the network, and $N_j \in A_i$ sends N_i the information on a set of paths, $SP^{t_0}_{jd\backslash i}$, as the optimal paths from N_j to N_d without passing through N_i during some time period $[t_0, t_1)$. Suppose, at time t_1, N_j sends N_i a different set of optimal paths from N_j to $N_d, SP^{t_1}_{jd\backslash i}$, due to some delay changes/component failures in the network. Then, $P^{t_0}_{jd\backslash i} \in SP^{t_0}_{jd\backslash i}$ is said to be a *faulty path* (or *faulty segment of a path*) known to N_i if and only if the following two conditions hold.

C1: $P^{t_0}_{jd\backslash i}$ does not contain any other faulty subpaths known to N_i. (Initially, assume there is no known faulty path in the network.)

C2: $D(P^{t_0}_{jd\backslash i}) < D(P^{t_1}_{jd\backslash i})$ or $D(P^{t_0}_{jd\backslash i}) = D(P^{t_1}_{jd\backslash i})$ and $n^{t_0}_{jd\backslash i} > n^{t_1}_{jd\backslash i}$, where $D(P)$ is the delay of a path P. □

The proposed strategy uses three tables to store the necessary information in order to route messages correctly. Given below are the detailed descriptions of these three tables NT', NT'', NT, and the notation for them and routing messages. Note that NT will be the routing table that a node actually uses to route packets in our strategy. However, we will also introduce two other tables NT' and NT'' since NT is derived from NT' and NT''.

Notation for Routing Messages RM: The routing message sent from N_j to $N_i \in A_j$ is a list of records (derived from N_j's routing table), one for each destination, and is denoted by $RM_{i \leftarrow j,d}$ for destination node N_d. A record is composed of two fields: 1) the delay of an optimal path from N_j to N_d without passing through N_i, denoted by $RM_{i \leftarrow j,d}.dly$, and 2) the number of paths with delay $RM_{i \leftarrow j,d}.dly$ from N_j to N_d, denoted by $RM_{i \leftarrow j,d}.num$.

Notation for NT': The table NT' of N_i is an array of records, each corresponding to a pair of $N_j \in A_i$ and a destination node $N_d \neq N_i$ in the network. That is, NT' has an entry/record $NT'_{i/j,d}$ with two fields: 1) $NT'_{i/j,d}.dly =$ the minimum delay from N_j to N_d without passing through N_i, and 2) $NT'_{i/j,d}.num =$ the number of paths with delay $NT'_{i/j,d}.dly$. Under the normal condition (no component failure/delay change), a node's NT' is used to store the routing messages from its neighbors.

When N_i receives the routing message from one of its neighbors, this information is stored in the corresponding entry of NT'. After the node receives all the routing messages during the current routing-message exchange interval, it uses another table NT'' (to be described below) to check whether each record of the message contains a known faulty path or not. (A record may correspond to multiple paths since there may be more than one optimal path between any given two nodes.) If the record does not contain any known faulty path, this record is ready to be used for generating new routing tables. If all the paths specified in the entry contain known faulty paths, then the corresponding $NT'_{i/j,d}.dly$ is set to ∞ and $NT'_{i/j,d}.num$ to 0. Otherwise, some (but not all) paths specified in the entry contain known faulty paths; in such a case, the delay is stored in the corresponding $NT'_{i/j,d}.dly$, and $NT'_{i/j,d}.num$ is set to the number of paths which do not contain any known faulty path segment.

Initially, N_i knows only the information about its neighbors, thus setting $NT'_{i/j,j}.dly := 0$, $NT'_{i/j,j}.num := 1$, $\forall N_j \in A_i$, and $NT'_{i/j,d}.dly := \infty, NT'_{i/j,d}.num := 0$ if $d \neq j, \forall N_j \in A_i$.

Notation for NT'': The structure of NT'' is exactly the same as NT', containing a record for each pair of a node in A_i and a destination node in the network. As mentioned above, NT'' is used to store the most recent information about the known faulty paths, and the information is used to check the validity (whether to contain a known faulty path or not) of routing messages from its neighbors before using these routing messages to update the three routing tables. Each entry of NT'' contains the most recent information on a known faulty path of the corresponding entry of NT'.

NT'' is updated in the following two cases. For $N_j \in A_i$:

Case 1: $RM_{i\leftarrow j,d}.dly > NT'_{i/j,d}.dly$. That is, the optimal path from N_j to N_d has been altered, and the delay from N_j to N_d increased.

Case 2: $RM_{i\leftarrow j,d}.dly = NT'_{i/j,d}.dly$, but $RM_{i\leftarrow j,d}.num < NT'_{i/j,d}.num$. That is, some of the optimal paths from N_j to N_d became invalid.

In both cases, $NT'_{i/j,d}.dly$ and $NT'_{i/j,d}.num$ are copied into the corresponding entry of NT'' in order to update information on known faulty paths.

Remark: In case $RM_{i\leftarrow j,d}.dly < NT'_{i/j,d}.dly$, we do not update the corresponding entry of NT'' because looping always results from known faulty paths. But in this case, the old path is not a known faulty path. Note that this case implies, in general, the recovery of a component failure or a temporary congestion.

Initially, $NT''_{i/j,d}.dly := \infty$, and $NT''_{i/j,d}.num := 0$, where $N_j \in A_i$ and $N_d \neq N_i$ is an arbitrary node in the network.

Notation for NT: NT is similar to a delay table under APRS, except NT now contains no loop. Each entry of NT has only one field, $NT_{i/j,d}.dly$, which is the minimum delay from N_i via $N_j \in A_i$ to N_d without going through any loop, so

$$NT_{i/j,d}.dly = NT'_{i/j,d}.dly + DL_{i,j}. \qquad (3)$$

Under the proposed strategy, the values of $NT_{i/j,d}.dly, \forall N_j \in A_i$ are sorted in ascending order, and are used to route messages and construct the routing messages, $RM_{j\leftarrow i,d}, \forall N_j \in A_i$; N_i will choose the minimum-delay path in NT as the optimal path to N_d and send N_j this path as its routing message, provided this path does not contain N_j.

After initializing NT', NT can be derived using (3).

Before formally stating our algorithm, let us consider the following two examples.

Example 1: This is an example of constructing the routing message $RM_{j\leftarrow i,d}$ under a stable condition, i.e., no known faulty paths are involved. Note that a path may become faulty when a link or node fails and/or the delay of a link increases due to the dynamic change of the network load. The next example will consider the case which involves known faulty paths. Assume $N_k, N_j, N_x \in A_i$, and N_i has knowledge of $n_{kd}, n_{kj}, n_{jd}, n_{xd}, n_{xj}, d_{kd}, d_{kj}, d_{jd}, d_{xd}, d_{xj}$ which correspond to $NT'_{i/k,d}.num, NT'_{i/k,j}.num, NT'_{i/j,d}.num, NT'_{i/x,d}.num, NT'_{i/x,j}.num, NT'_{i/k,d}.dly, NT'_{i/k,j}.dly, NT'_{i/j,d}.dly, NT'_{i/x,d}.dly, NT'_{i/x,j}.dly$, respectively. Since we assume no known faulty paths involved here, all numbers are consistent with each other, and NT'' can be ignored in this example.

Then, as shown in Fig. 3, N_i is aware of the following three paths to N_d:

path 1 via N_j with delay $d_{jd} + DL_{i,j} = NT_{i/j,d}.dly$,
path 2 via N_k with delay $d_{kd} + DL_{i,k} = NT_{i/k,d}.dly$,

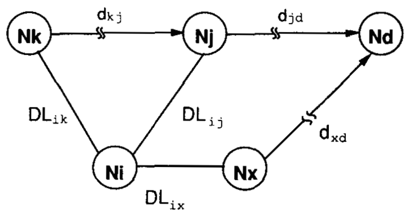


Fig. 3. Example 1.

path 3 via N_x with delay $d_{xd} + DL_{i,x} = NT_{i/x,d}.dly$,

Suppose $d_{jd} + DL_{i,j} < d_{kd} + DL_{i,k} < d_{xd} + DL_{i,x}$, and $d_{xd} \neq d_{xj} + d_{jd}$. That is, the optimal path from N_i to N_d is $N_i \rightarrow N_j \rightarrow \cdots \rightarrow N_d$, and the optimal path from N_x to N_d does not pass through N_j.

When constructing N_i's routing message $RM_{j\leftarrow i,d}$ for N_j which contains all optimal paths to N_d, we need to consider the following three cases. (Among these three cases, path 1 will never be chosen by N_i as an optimal path because it will lead to a ping-pong type loop [12].)

Case 1: If $d_{kd} \neq d_{kj} + d_{jd}$, then path 2 will be sent to N_j as the optimal path from N_i to N_d because path 2 is the minimum-delay loop-free path from N_i to N_d without passing N_j.

Case 2: If $d_{kd} = d_{kj} + d_{jd}$ and $n_{kd} = n_{kj}n_{jd}$, then path 3 will be sent to N_j as the optimal path from N_i to N_d because path 2 will lead to the (source) loop, $N_j \rightarrow N_i \rightarrow N_k \rightarrow \cdots \rightarrow N_j \rightarrow \cdots \rightarrow N_d$.

Case 3: If $d_{kd} = d_{kj} + d_{jd}$ and $n_{kd} > n_{kj}n_{jd}$, then path 2 will be sent as the optimal path, but the number of optimal paths will be changed from n_{kd} to $n_{kd} - n_{kj}n_{jd}$. □

Since the routing information from N_j and N_k may be derived at different times, the two nodes' knowledge of d_{jd} may be different. This problem can be handled by Procedure B (to be described later) which removes all faulty paths. As N_i is a neighbor of N_j, it will always be informed of the change of d_{jd} by N_j directly. At the same time, the d_{kd} information from N_k may still include the old d_{jd} (assume that the optimal path from N_k to N_d runs through N_j), i.e., the change of d_{jd} has not propogated to N_i via N_k. Therefore, there are two cases[1]: either d_{jd} increases or decreases. If d_{jd} decreases, the path via N_k will not be selected since d_{kd} must be larger than the new d_{jd}. Thus, the optimal path will be chosen to be path 1 or path 3.

If d_{jd} increases, the old d_{jd} will be saved in NT'', and used by Procedure B to remove any possible known faulty path. Example 2 shows how the tables are updated and routing messages constructed when d_{jd} increases.

Example 2: This example illustrates the process of updating delay tables and constructing $RM_{j\leftarrow i,d}$ upon N_i's receipt of a routing message which includes a new set of optimal paths. The optimal path from N_j to N_d in Example 1 becomes invalid due to some link failure (Fig. 4), and N_j detects this failure at time t_0.

[1]There is, in fact, a third case, where d_{jd} does not change, but n_{jd} changes. We will consider all three cases formally in Theorem 1.

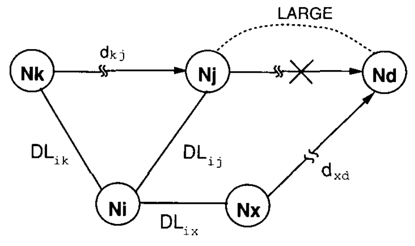


Fig. 4. Example 2.

Upon receiving the new set of optimal paths from N_j to N_d with delay larger than d_{jd} at time $t_1 = t_0 + 1, N_i$ will store the old set of optimal paths (N_i's known faulty paths), d_{jd} and n_{jd} in NT'', and then record the new paths in NT'. As in Example 1, there are three cases to consider.

In Case 1, because the optimal path running through both N_k and N_x does not pass through N_j, path 2 will still be sent to N_j as the optimal path. However, in Case 2, during $[t_1, t_f)$ where t_f is the time N_k detects the link failure, N_i can use the information in NT'' to learn that the path via N_k will pass through N_j, i.e., it contains a known faulty segment $N_j \to \cdots \to N_d$. Thus, N_i will not use the routing message received from N_k to update its routing table directly; instead, it will set $NT'_{i/k,d}.dly := \infty$ and $NT'_{i/k,d}.num := 0$. This path will be ignored until a new path is found. As a result, during this time interval, path 3 will be sent to N_j as the optimal path from N_i to N_d. In our algorithm, LARGE (in Fig. 4, which is the delay of the new optimal path from N_j to N_d), d_{jd}, n_{jd}, d_{kj}, and n_{kj}, are recorded in $NT'_{i/j,d}.dly, NT''_{i/j,d}.dly, NT''_{i/j,d}.num, NT'_{i/k,j}.dly$, and $NT'_{i/k,j}.num$, respectively; d_{kd} and n_{kd} are found from $RM_{i\leftarrow k,d}.dly$ and $RM_{i\leftarrow k,d}.num$, respectively.

In Case 3, N_i will set $NT'_{i/k,d}.dly := d_{kd}$, but the number of optimal paths will be changed to $NT'_{i/k,d}.num := n_{kd} - n_{kj}n_{jd}$. Note that $n_{kj} = NT'_{i/k,j}.num$ and $n_{jd} = NT''_{i/j,d}.num$. □

Given below is the proposed algorithm for an arbitrary node N_i. In addition to an initialization procedure, it contains two procedures: Procedure A for constructing the routing messages to its neighbors, and Procedure B for updating its own routing tables.

Procedure Initialization: For all $N_j \in A_i$ **do**

```
{
$NT_{i/j,j}.dly := DL_{i,j}; NT'_{i/j,j}.dly := 0; NT'_{i/j,j}.num := 1;$
$NT''_{i/j,j}.dly := \infty; NT''_{i/j,j}.num := 0$
}
```

Set all other $.dly$ entries to ∞, and $.num$ entries to 0.

For other nodes which are not adjacent to N_i, there should be no corresponding entries in NT, NT', and NT'' of N_i. However, they will be added in all three tables whenever their information reaches N_i.

Procedure A: Construct routing message $RM_{j\leftarrow i,d}$ for each $N_j \in A_i$ and N_d in the network, and maintain NT and NT'.

1) $\forall N_k \in A_i, k \neq j$, sort $NT_{i/k,d}.dly$ in ascending order into a list L.

2) If L is empty, or the head of L is ∞, then $RM_{j\leftarrow i,d}.dly := \infty; RM_{j\leftarrow i,d}.num := 0$.

3) Let the set of smallest entries of L be S, which may contain more than one element.

```
$RM_{j\leftarrow i,d}.num := 0;$
For all $NT_{i/k,d}.dly \in S$ do
{
if $(NT'_{i/k,d}.dly \leq NT'_{i/j,d}.dly)$
    $RM_{j\leftarrow i,d}.num := RM_{j\leftarrow i,d}.num + NT'_{i/k,d}.num;$
else if $(NT'_{i/k,d}.dly - NT'_{i/j,d}.dly \neq NT'_{i/k,j}.dly)$
    $RM_{j\leftarrow i,d}.num := RM_{j\leftarrow i,d}.num + NT'_{i/k,d}.num;$
else if $(NT'_{i/k,d}.num \neq NT'_{i/j.d}.num \times NT'_{i/k,j}.num)$
    $RM_{j\leftarrow i,d}.num := RM_{j\leftarrow i,d}.num + NT'_{i/k,d}.num - NT'_{i/j,d}.num \times NT'_{i/k,j}.num$
}
if $(RM_{j\leftarrow i,d}.num > 0)$
    $RM_{j\leftarrow i,d}.dly := NT_{i/k,d}.dly;$
else
    $L := L - S$; go to Step 2;
```

Procedure B: Update routing tables. If some destination node N_d specified in the routing messages has not been recorded in all of NT, NT', and NT'', then we add new entries $(NT'_{i/k,d}.num, NT'_{i/k,d}.dly, NT''_{i/k,d}.num, NT''_{i/k,d}.dly, NT_{i/k,d}.dly$ for all $N_k \in A_i)$ to the three tables and set all other $.dly$ entries to ∞ and $.num$ entries to 0; this is for the case when some node could not be reached before or a new node is added to the network.

When N_i receives $RM_{i\leftarrow k,d}$, this message is used to update $NT'_{i/k,d}.dly, NT'_{i/k,d}.num, NT''_{i/k,d}.dly, NT''_{i/k,d}.num$, and $NT_{i/k,d}.dly$ as follows (see Example 2).

```
1) Update $NT''$.
  if $(RM_{i\leftarrow k,d}.dly > NT'_{i/k,d}.dly)$ {
      $NT''_{i/k,d}.dly := NT'_{i/k,d}.dly;$
      $NT''_{i/k,d}.num := NT'_{i/k,d}.num$ :
      }                                                    (a)
  }
  else if $((RM_{i\leftarrow k,d}.dly = NT'_{i/k,d}.dly)$ and $(RM_{i\leftarrow k,d}.num < NT'_{i/k,d}.num))$ {
      if $(NT''_{i/k,d}.dly = NT'_{i/k,d}.dly)$
          $NT''_{i/k,d}.num := NT''_{i/k,d}.num + NT'_{i/k,d}.num - RM_{i/k,d}.num;$      $(b')$
      else {
          $NT''_{i/k,d}.dly := NT'_{i/k,d}.dly$ :
          $NT''_{i/k,d}.num := NT'_{i/k,d}.num$
          $-RM_{i/k,d}.num;$                                 (b)
  }}

2) Update $NT'$ and $NT$.
  $NT'_{i/k,d}.dly := RM_{i\leftarrow k,d}.dly$
  $NT'_{i/k,d}.num := RM_{i\leftarrow k,d}.num$
  if $(NT'_{i/k,d}.num \neq 0)$ {
      For each $N_j \in A_i \backslash \{N_k\}$ do {
          if $(NT'_{i/k,d}.dly = NT'_{i/k,j}.dly + NT''_{i/j,d}.dly)$ {
                                                             (c)
              $NT'_{i/k,d}.num := NT'_{i/k,d}.num$
                  $-NT'_{i/k,j}.num \times NT''_{i/j,d}.num;$     (d)
```

```
            if ($NT'_{i/k,d}.num = 0$) {
                  $NT'_{i/k,d}.dly := \infty$;              (e)
                  break;
   }}}}
   $NT_{i/k,d}.dly := NT'_{i/k,d}.dly + DL_{i,k}$;          □
```

Since the routing messages from different neighbors may arrive at different routing message exchange intervals, most operations of Step 2 in Procedure B (except for the first two copying statements) have to be delayed until all routing messages arrive. As we assume that a node can detect the failure of its own neighbors and/or its own links, the node will not wait for the routing message through a broken link/node. So, Procedure B can be executed at appropriate times and all NT'''s are up to date when a node executes Procedure B.

Note that although the k in Procedure B refers to all neighbors $N_k \in A_i$, Part 1) of Procedure B will actually run only for those nodes that report troubles since only such nodes can satisfy one of the two if-statements. Similarly, Part 2) of Procedure B will only affect those nodes (N_k's) with property A, when the node (N_j) that enabled N_k's to possess property A reports troubles (statements (c) in Procedure B.)

Suppose N_i's delay tables, NT and NT', contain no known faulty paths of N_i (this property can be achieved by Procedure B); then the implementation of Procedure A is straightforward. Since $NT'_{i/k,d}.dly$ is the delay of the optimal paths from N_k to N_d and $NT'_{i/j,d}.dly$ is the delay of the optimal paths from N_j to N_d, if $NT'_{i/k,d}.dly \leq NT'_{i/j,d}.dly$, or if $NT'_{i/k,d}.dly > NT'_{i/j,d}.dly$ and $NT'_{i/k,d}.dly - NT'_{i/j,d}.dly \neq NT'_{i/k,j}.dly$, then one of the optimal paths from N_i via N_k to N_d will not pass through N_j. Otherwise, $NT'_{i/k,d}.dly > NT'_{i/j,d}.dly, NT'_{i/k,d}.dly - NT'_{i/j,d}.dly = NT'_{i/k,j}.dly$, and $NT'_{i/k,d}.num - NT'_{i/k,j}.num \times NT'_{i/j,d}.num$ is the number of optimal paths from N_k to N_d that do not pass through N_i and N_j. Note that $NT'_{i/k,d}.num - NT'_{i/k,j}.num \times NT'_{i/j,d}.num$ is not necessarily the number of loop-free optimal paths from N_i via N_k to N_d since there could be more than one optimal path from N_i to N_k.

As proven below, Procedure B removes all known faulty paths in the network delay tables, provided the tables did not previously contain any such path.

Theorem 1: Suppose there are no known faulty paths of N_i in its current delay table NT'. If this condition holds for every node in the network, then the condition will still hold for every node in the network, even after executing the proposed updating process.

Proof: Recall the definition of a known faulty path and consider Fig. 5. We only need to consider the behavior of our algorithm after a link failure (delay change). Since if no failure/change occurs then there is no faulty path at all, Procedure A is sufficient to make all paths specified by the network delay table loop-free. Since Step 1) of Procedure B will not be executed if no failure/change occurs, Step 2) of Procedure B degenerates itself to just a process of copying the routing messages into NT', i.e., $NT''_{i/j,d}.dly := \infty$ in (c) of Procedure B.

Recall the situation shown in Fig. 2 where a link failure had occurred at time t_0; N_j detected this failure at time $t_1 > t_0$,

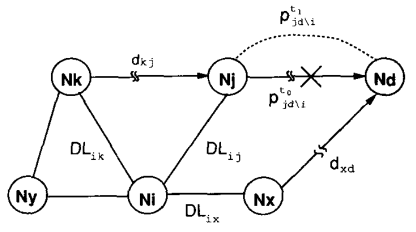


Fig. 5. Theorem 1.

and N_k (a representative node of S_i) detected it at time $t_2 > t_1$. If an optimal path from N_k to N_d does not pass through N_j (like N_x in Figs. 2 and 5), the failure in $P^{t_0}_{jd\backslash i}$ will not affect this optimal path. As mentioned above, Step 1) of Procedure B will not be executed, and Step 2) of Procedure B degenerates itself to a copying process. Hence, we will focus only on the situation when $N_k \rightarrow \cdots \rightarrow N_j \rightarrow \cdots \rightarrow N_d$ is the optimal path from N_k to N_d without passing through N_i.

Part 1: Until $t_1 + 1, N_i$ does not know anything about the fault in $P^{t_0}_{jd\backslash i}$, which is the current optimal path from N_i via N_j to N_d.

Part 2: At time $t_1 + 1, N_j$ will inform N_i of the failure in $P^{t_0}_{jd\backslash i}$ by sending N_i a new set of paths, $SP^{t_0}_{jd\backslash i}$, while all other nodes still send N_i the obsolete set of paths, $SP^{t_0}_{kd\backslash i}$. Thus, only one entry of $NT', NT'_{i/j,d}.dly$, is copied into the corresponding entry of NT'' by Step 1) of Procedure B. $NT''_{i/j,d}.num$ will be set to the number of the faulty paths with delay $NT'_{i/j,d}.dly$ by the statements (a) and (b) or (b′) in Step 1) of Procedure B. In Step 2) of Procedure B, all $NT'_{i/k,d}.dly$'s will be checked by statement (c) to see if the entry $NT'_{i/k,d}$ contains any known faulty path [$NT''_{i/j,d}.dly$ in statement (c)]. If it contains the faulty segment of $P^{t_0}_{jd\backslash i}, NT'_{i/k,j}.num \times NT''_{i/j,d}.num$ (which is the number of paths containing this known faulty segment in $NT''_{i/j,d}.dly$) will be subtracted from $NT'_{i/k,d}.num$ [statement (d)]. Moreover, if $NT'_{i/k,d}.num = 0$, i.e., all paths with delay $NT'_{i/k,d}.dly$ contain known faulty segments, set $NT'_{i/k,d}.dly := \infty$. Therefore, the delay table NT' of N_i contains no faulty path known so far.

Part 3: During the time period $[t_1 + 2, t_2 + 1), N_j$ may send N_i another new set of paths $SP^{t_2}_{jd\backslash i}$ whenever it finds a path with delay $\leq D(P^{t_1}_{jd\backslash i})$. Thus, $D(P^{t_2}_{jd\backslash i}) < D(P^{t_1}_{jd\backslash i})$, or $D(P^{t_2}_{jd\backslash i}) = D(P^{t_1}_{jd\backslash i})$ and $n^{t_2}_{jd\backslash i} > n^{t_1}_{jd\backslash i}$. However, in both cases, $NT''_{i/j,d}.dly$ and $NT''_{i/j,d}.num$ will not be overwritten by Step 1) of Procedure B, i.e., the set of known faulty paths, $SP^{t_0}_{jd\backslash i}$, is still in $NT''_{i/j,d}.dly$ and $NT''_{i/j,d}.num$, although $NT'_{i/j,d}.dly$ and $NT'_{i/j,d}.num$ may be changed during this time period. Since $NT''_{i/j,d}.dly$ and $NT''_{i/j,d}.num$ are not changed, and N_k still sends N_i the obsolete set of paths $SP^{t_0}_{kd\backslash i}, NT'_{i/k,d}.dly$ and $NT'_{i/k,d}.num$ are set to the same value as in Part 2.

Part 4: At time $t_2 + 1$ (there can be many different t_2's, corresponding to different N_k's and different paths from N_j to N_k), the procedure of handling the routing message $RM_{i \leftarrow j,d}$ is the same as Part 3. However, N_k will send N_i a new set of valid paths (for destination N_d), $SP^{t_1}_{kd\backslash i}$. As derived from

Parts 2 and 3, there are two possible causes of $NT'_{i/k,d}.dly$ and $NT'_{i/k,d}.num$ at this time.

Case 1: When $NT'_{i/k,d}.dly = \infty$ and $NT'_{i/k,d}.num = 0$, i.e., no valid path exists. Step 1) of Procedure B will not be executed because $RM_{i\leftarrow k,d}.dly < NT'_{i/k,d}.dly = \infty$.

Case 2: When $NT'_{i/k,d}.dly$ is equal to both $D(P^{t_0}_{kd\backslash i})$ and $D(P^{t_1}_{kd\backslash i})$, and $NT'_{i/k,d}.num$ is the number of paths with delay $NT'_{i/k,d}.dly$ after removing those paths containing known faulty segments (derived in Part 2). In this case, $RM_{i\leftarrow k,d}.dly$ must be equal to $NT'_{i/k,d}.dly$ because there are still some valid paths with delay $NT'_{i/k,d}.dly$, and $NT'_{i/k,d}.dly$ is the minimum delay for the path from N_i via N_k to N_d. Moreover, $RM_{i\leftarrow k,d}.num > NT'_{i/k,d}.num$ because N_k knows at least $NT'_{i/k,d}.num$ paths with delay $RM_{i\leftarrow k,d}.dly$. Thus, according to our algorithm, Step 1) of Procedure B will not be executed either. Those paths containing known faulty segments will be removed by Step 2) of Procedure B. Therefore, if there is some other node $N_y \in S_i$, which sends N_i the routing message containing the information of a path $N_y \rightarrow \cdots \rightarrow N_k \rightarrow \cdots \rightarrow N_j \rightarrow \cdots \rightarrow N_d$ in Step 2) of Procedure B, our algorithm will only subtract the number of paths that pass through N_j from $NT'_{i/y,d}.num$ because the faulty segment $N_k \rightarrow \cdots \rightarrow N_j \rightarrow \cdots \rightarrow N_d$ is not figured in $NT''_{i/k,d}.dly$ and $NT''_{i/k,d}.num$ in both cases. That is, Step 2) of Procedure B removes each known faulty path exactly once. Consequently, Step 2) of Procedure B can remove all known faulty paths of N_i.

Part 5: After $t_2 + 1$, the procedure of handling the routing message $RM_{i\leftarrow j,d}$ is the same as Part 3. The routing message $RM_{i\leftarrow k,d}$ from N_k can be processed in the same way as in Part 4. □

Corollary 1: If there are no loops and no known faulty paths in the delay tables of all nodes in the entire network during some time interval $[m, m+1)$, then under our strategy, this property will hold in the next time interval $[m+1, m+2)$.

Proof: Since if there are no loops and no known faulty paths in all delay tables, our algorithm can detect and avoid all source loops, and generate the correct number of optimal paths between any two nodes. So, all delay tables are still loop-free during the next time interval. Thus, by Theorem 1, our algorithm can remove all known faulty paths from neighbor nodes' routing messages during the next time interval. □

Since there are no loops and no known faulty paths upon initialization of a network, by Corollary 1 the network will always stay loop-free under our strategy.

As mentioned above, our algorithm, which is a modified version of APRS, is actually an order-one strategy, only adding the number of paths with the optimal delay to routing messages, and is loop-free in case of link failures and/or network structural changes. The performance of our strategy will be analyzed in Section IV; examples and comparison between APRS and the strategy in [13] will also be given there.

III. Analysis of Operational Overhead

Let the network have n nodes and $|A_i| \leq a, \forall i \in \{0, \cdots, n\}$, where $|A_i|$ is the node degree of N_i. Then, the proposed algorithm induces the space and time overheads as discussed below.

Space Requirement: The algorithm needs three tables NT, NT', and NT''. NT has at most na entries, and both NT' and NT'' have at most $2na$ entries. $O(na)$ memory locations are thus needed for these tables. Moreover, the size of a routing message is $O(na)$ since each record/entry in the message contains two fields. Therefore, the proposed algorithm has $O(na)$ space complexity.

Time Complexity: The proposed algorithm consists of three procedures: Procedure A, Procedure B, and the procedure for sending the routing messages to adjacent nodes.

- Procedure A essentially contains two loops. The outer loop is used to construct the routing message for each destination node. Thus, they will execute once for each destination, i.e., n times per unit time for the entire network. In the outer loop, Step 1) is a sorting procedure, so its complexity is $O(a!)$. Steps 2) and 3) form the inner loop; basically, it will execute a times for each iteration of the outer loop. Thus, Procedure A has complexity $O(na!)$. However, if all $NT_{i/k,d}$'s are not changed, then we do not have to sort the list each time, construct new $RM_{j\leftarrow i,d}$, and send the corresponding routing message. Moreover, this is usually the case because the network does not change frequently. Thus, the complexity of Procedure A is usually much lower than $O(na!)$.
- Procedure B stores new routing messages and computes new entries of the network delay tables. It contains two steps. The first step is used to update NT''. According to our algorithm, this step is $O(na)$. The second step is used to update NT and NT'. In addition to storing new routing messages in NT', it has a loop for removing known faulty paths. In a brute-force implementation, updating each entry needs $O(a)$ time, thus needing $O(na^2)$ time for updating all tables. However, with a careful implementation, a node maintains a link list for each destination node whose components are pointers to each NT'' entry containing a faulty path. When updating an NT'' entry in Step 1), we move that entry to the head of the list to avoid the corresponding search. Therefore, updating an entry needs only a constant time, and updating the entire table needs $O(na)$ time. Since both steps need $O(na)$ time, Procedure B is an $O(na)$ procedure.
- Sending procedure totally sends $O(a)$ messages, each with size $O(n)$, thus needing $O(na)$ time.

The worst-case time complexity of our algorithm is $O(na!)$. Since a is usually a small constant (less than 10), it is an $O(n)$ algorithm, thus making our algorithm particularly suitable for sparse networks.

IV. Performance Analysis and Demonstrative Examples

As mentioned earlier, the information in the delay tables may become obsolete due to component failures and structural changes which are not known immediately to the source node N_s. In such a case, the $OP_{s,d}(m)$ derived from N_s's routing table may *not* be the actual optimal path. In the examples and analyses that will follow, $OP_{s,d}$ denotes the "current" actual

TABLE I
DELAY TABLES OF N_1, N_2, N_3, N_4 TO DESTINATION NODE N_5 UNDER OUR ALGORITHM

neighbor node	Time				
	0	1	2	3	4
Delay Table of N_1 to destination N_5					
N_2	7/3/1/∞/0	7/3/1/∞/0	27/23/1/3/1	27/23/2/3/1	27/23/2/3/1
N_3	9/4/1/∞/0	9/4/1/∞/0	25*/20/1/4/1	25/20/1/4/1	25/20/1/4/1
Delay Table of N_2 to destination N_5					
N_1	13/9/1/∞/0	∞/∞/0/∞/0	∞/∞/0/∞/0	29/25/1/∞/0	29/25/1/∞/0
N_3	7/4/1/∞/0	∞/∞/0/∞/0	23/20/1/∞/0	23/20/1/∞/0	23/20/1/∞/0
N_4	3/2/1/∞/0	23*/22/1/2/1	23/22/1/2/1	23/22/1/2/1	23/22/1/2/1
Delay Table of N_3 to destination N_5					
N_1	12/7/1/∞/0	∞/∞/0/∞/0	∞/∞/0/∞/0	∞/∞/0/∞/0	∞/∞/0/∞/0
N_2	6/3/1/∞/0	∞/∞/0/∞/0	26/23/1/∞/0	∞/∞/0/23/1	∞/∞/0/23/1
N_4	4/2/1/∞/0	∞/∞/0/2/1	∞/∞/0/2/1	∞/∞/0/2/1	∞/∞/0/2/1
N_5	20/0/1/∞/0	20*/0/1/∞/0	20/0/1/∞/0	20/0/1/∞/0	20/0/1/∞/0
Delay Table of N_4 to destination N_5					
N_2	∞/∞/0/29/1	∞/∞/0/29/1	∞/∞/0/29/1	24/23/1/29/1	24/23/1/29/1
N_3	22*/20/1/∞/0	22/20/1/∞/0	22/20/1/∞/0	22/20/1/∞/0	22/20/1/∞/0
N_5	∞/∞/0/0/1	∞/∞/0/0/1	∞/∞/0/0/1	∞/∞/0/0/1	∞/∞/0/0/1

TABLE II
DELAY TABLES OF N_1, N_2, N_3, N_4 TO DESTINATION NODE N_5 UNDER APRS

neighbor nodes entry	Time										
	$t \in (-\infty, 0)$	0	1	2	3	4 − 15	16	17	18	19	20
Delay Table of N_1 to destination N_5											
N_2	7	7	7	9	9	$\lfloor \frac{t}{2} \rfloor \times 2 + 7$	23	23	25	25	27
N_3	9	9	9	11	11	$\lfloor \frac{t}{2} \rfloor \times 2 + 9$	25	25	25	25	25*
Delay Table of N_2 to destination N_5											
N_1	11	11	11	11	13	$\lceil \frac{t}{2} \rceil \times 2 + 9$	25	27	27	29	29
N_3	7	7	7	9	9	$\lfloor \frac{t}{2} \rfloor \times 2 + 7$	23	23	23	23	23
N_4	3	3	5	5	7	$\lfloor \frac{t}{2} \rfloor \times 2 + 3$	19	21	21	23*	23
Delay Table of N_3 to destination N_5											
N_1	12	12	12	12	14	$\lceil \frac{t}{2} \rceil \times 2 + 10$	26	28	28	30	30
N_2	6	6	6	8	8	$\lfloor \frac{t}{2} \rfloor \times 2 + 6$	22	22	24	24	26
N_4	4	4	6	6	8	$\lceil \frac{t}{2} \rceil \times 2 + 4$	20	22	22	24	24
N_5	20	20	20	20	20	20	20	20*	20	20	20
Delay Table of N_4 to destination N_5											
N_2	4	4	4	6	6	$\lfloor \frac{t}{2} \rfloor \times 2 + 4$	20	20	22	22	24
N_3	6	6	6	8	8	$\lfloor \frac{t}{2} \rfloor \times 2 + 6$	22	22	22	22	22*
N_5	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

TABLE III
DELAY TABLES OF N_1, N_2, N_3, N_4 TO DESTINATION NODE N_5 UNDER THE THIRD-ORDER (MINIMAL ORDER LOOP-FREE) ROUTING STRATEGY

neighbor nodes entry	Time					
	$time \in (-\infty, 0)$	$time = 0$	$time = 1$	$time = 2$	$time = 3$	$time = 4$
Delay Table of N_1 to destination N_5						
N_2	7	7	7	11	27	27
N_3	9	9	9	11	25*	25
Delay Table of N_2 to destination N_5						
N_1	13	13	13	13	∞	29
N_3	7	7	7	23	23*	23
N_4	3	3	23	23	23*	23
Delay Table of N_3 to destination N_5						
N_1	12	12	12	12	∞	∞
N_2	6	6	6	∞	∞	∞
N_4	4	4	∞	∞	∞	∞
N_5	20	20	20	20*	20	20
Delay Table of N_4 to destination N_5						
N_2	∞	∞	∞	24	24	24
N_3	22	22*	22	22	22	22
N_5	2	∞	∞	∞	∞	∞

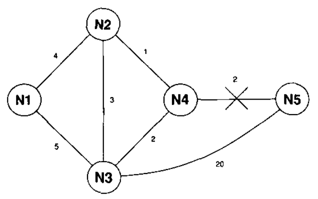


Fig. 6. Example 3.

optimal path from N_s to N_d and $DOP_{s,d}$ denotes the delay of $OP_{s,d}$. Before analyzing the performance, let us consider an illustrative example.

Example 3: This example illustrates the network operations under our algorithm in Table I after the failure of link $L_{4,5}$ is detected at time 0 (Fig. 6). Each entry of the table contains five numbers which are in the order of $NT_{i/j,d}.dly/NT'_{i/j,d}.dly/NT'_{i/j,d}.num/NT''_{i/j,d}.dly/NT''_{i/j,d}.num$. A $*$ symbol marks the entry where the optimal path is found for the first time after the link failure, and an ∞ symbol indicates paths with an infinite delay or paths with a loop. At the time of failure $t = 0, N_4$ can determine the new optimal path to N_5 from its routing table since the path through N_3 does not use the failed link. At time $t = 1, N_3$ can determine its new optimal path to N_5 after receiving N_4's routing message with an ∞ delay and rejecting the paths through N_1 and N_2 since both contain the known faulty path $N_4 \rightarrow \cdots \rightarrow N_5$. In the same manner, N_2 and N_1 can determine their new optimal paths to N_5 at time $t = 1$ and $t = 2$, respectively. Tables II and III are the operations under APRS and the third-order strategy which is shown in [13] to be the minimal order loop-free strategy for the network in Fig. 6. As can be seen in Table I, N_1, N_2, N_3, N_4 need 2, 1, 1, 0, time units, respectively, to find new optimal paths to N_5 under the proposed algorithm, while they need 20, 19, 17, 20 time units under APRS in Table II, and 3, 3, 2, 0 time units under the third-order strategy in [13] in Table III. Obviously, our algorithm is far better than the others. □

Definition 2 [13]: The *hop function* $h: SP \rightarrow I^+$ is defined as the number of links in a path, where I^+ is the set of positive integers.

Lemma 1: An arbitrary node N_s can determine an optimal path to N_d in $h(OP_{s,d})$ time units after initialization.

Proof: By assumption, there are no loops and no known faulty paths in the network at the time of initialization. By Corollary 1, during the interval $(0, h(OP_{s,d})]$, there are still no loops and no known faulty paths in the network. Moreover, because routing messages are exchanged once per unit time, the information is broadcast exactly one hop per unit time. Thus, using the proposed algorithm, N_s can get the information from N_d in $h(OP_{s,d})$ time units. □

Since a node failure can be represented as the failure of all links attached to it, network structural changes can be represented as link-delay changes. Moreover, because a link failure can be viewed as its link delay becoming ∞, without loss of generality, we can consider only link-delay changes in the analysis. Assume that the delay of link $L_{i,j}$ had changed and this change was detected at time 0, while the delays of other links remain unchanged. We make a further assumption that there always exists at least one path from a node N_s to another node N_d. Otherwise, the corresponding delay in the delay table of N_s will simply be set to ∞.

In the following performance analysis, we need to consider the four cases distinguished by the relation between $L_{i,j}$ and $OP_{s,d}$.

Case 1: $L_{i,j} \in OP_{s,d}$ despite the decrease/increase of $DL_{i,j}$. $OP_{s,d}$ remains unchanged, even with the change in $DL_{i,j}$. N_s can obtain the correct value of $DOP_{s,d}$ in $h(OP_{s,i})$ time units since N_s is $h(OP_{s,i})$ hops away from $L_{i,j}$ whose delay has changed. □

Case 2: $L_{i,j}$ is not part of $OP_{s,d}$ regardless of the change of DL_{ij}. That is, $OP_{s,d}$ will not be affected by this link-delay change. □

Cases 1 and 2 are very simple, and their results and performances can be predicted easily because the change has no effects on the routing decision for the packets from N_s and N_d. In these two cases, our strategy has exactly the same performance as APRS and the strategy in [13] in case of link failures. However, the remaining two cases are much more complicated than these. It is necessary to introduce the following definitions before discussing them.

Definition 3: **Screen** of a path P, denoted by $scn(P)$ [12]. Let a path $P = (N_{i_0}, N_{i_1}, \cdots, N_{i_m})$, and the pair $(N_{i_k}, N_{i_{k+1}}) \equiv L_{i_k, i_{k+1}}$ is the first link in the ordered sequence representation of P, whose delay change is not known to N_{i_0}. Then, if $DL_{i_k, i_{k+1}}$ increases, then $scn(P) = k$, else $scn(P) = -k$. In case there is no link delay in P recently, $scn(P) = \infty$.

Definition 4: A new set of paths, $SP'_{s,d}$, is defined as $SP'_{s,d} \equiv SP_{s,d} - \{P | P \in SP_{s,d}$, and P contains a loop$\}$. Obviously, $SP'_{s,d} \subseteq SP_{s,d}$.

Case 3: • *$L_{i,j} \in OP_{s,d}, DL_{i,j}$ increased recently.*

• *$L_{i,j}$ is not part of $OP_{s,d}$ any longer after this change.*

In this case, all paths containing $L_{i,j}$ have a positive screen value. This is similar to those cases discussed in [12], [13] because a link failure can be treated as the delay of that link increased to ∞. So, the link cannot be part of any optimal path. Let m_c be the minimal time units required for a source node N_s to find its new optimal path to N_d in case of a link-delay change under our strategy, and let m_k be the minimal time units required for N_s to find its new nonfaulty optimal path under the kth order routing strategy in [12], [13]. Also, let r denote the delay of the new optimal path $OP_{s,d}$, and $SP^k_{s,d} \equiv SP_{s,d} - \{P | P \in SP_{s,d}$, and P contains a loop whose order is less than, or equal to, $k\}$. Clearly, $SP'_{s,d} \subseteq SP^k_{s,d} \forall k > 0$. The authors of [12], [13] proved the following relations:

$$m_0 = \max\{scn(P) | P \in SP_{s,d} \text{ and } d(P) < r\}$$
$$m_k = \max\{scn(P) | P \in SP^k_{s,d} \text{ and } d(P) < r\}$$
$$m_c \le \max\{scn(P) | P \in SP'_{s,d} \text{ and } d(P) < r\}$$

where APRS is order 0, and $m_c \le m_k \le m_0, \forall k > 0$.

Therefore, our strategy is better than *any* order routing strategy [13] and, of course, better than APRS.

Case 4: • *$L_{i,j}$ is not part of $OP_{s,d}$ and $DL_{i,j}$ decreases.*

• *This decrease makes $L_{i,j}$ become part of $OP_{s,d}$.*

In this case, some paths will have negative screen values, including the new optimal path. Because the new optimal path from N_s to N_d is $N_s \to \cdots \to N_i \to N_j \to \cdots \to N_d$, and $OP_{j,d}$ is already known to N_i, so $m_c = -scn(OP_{s,d}) = h(OP_{s,i})$. Since the information on the decreased $DL_{i,j}$ propagates through $OP_{s,i}$ one hop per unit time, $DOP_{s,i}$ and $DOP_{s,d}$ need $h(OP_{s,i})$ time units to propagate the information from N_i to N_s. □

The following observations indicate the superiority of our strategy to others.

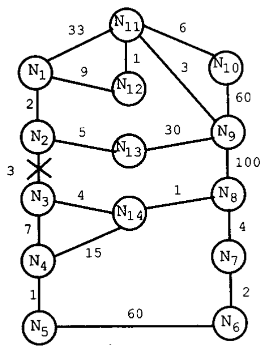


Fig. 7. Example 4.

1) In Cases 1 and 2, a link-delay change will not affect the message routing under our strategy. In Case 1, nodes can obtain the correct delay of an optimal path in $h(OP_{s,i})$ time units. This is optimal under any APRS-related routing strategies because the information is propagated one hop per unit time.

2) In Case 4, a source node can find a new nonfaulty optimal path to the destination after each link-delay change in $h(OP_{s,i})$ time units. This is also optimal under all APRS-related routing strategies for the same reason as above.

3) In Case 3, the performance of our strategy is better than, or at least the same as, that of APRS and any order strategy in [13].

Example 4: See Fig. 7.

The network in Fig. 7 is a part of the real APARNET. In order to make this example more illustrative, $DL_{2,3}$ and $DL_{8,9}$ are assigned to 3 and 100, respectively. Suppose $L_{2,3}$ fails at time 0. Basically, this network becomes two sets of nodes, S_1 and S_2, which are connected only via $L_{8,9}$ after this failure, where

$$S_1 = \{N_1, N_2, N_9, N_{10}, N_{11}, N_{12}, N_{13}\}$$
$$S_2 = \{N_3, N_4, N_5, N_6, N_7, N_8, N_{14}\}.$$

Obviously, the optimal paths between two nodes within the same set are not affected by this failure, but the optimal paths between two nodes which belong to different sets are all changed. Moreover, as can be seen in Fig. 7, an arbitrary node, say $N_1 \in S_1$, will find its optimal paths to any arbitrary node in S_2 in the same amount of time because all the *screen* values of the paths from N_1 to any node in S_2 are identical. Similarly, an arbitrary node in S_2 can find its optimal paths to any arbitrary nodes in S_1 in the same amount of time. Table IV shows the operations of all nodes in S_1 to find new nonfaulty optimal paths to N_3 after the failure of link $L_{2,3}$, where N_3 is a representative node for S_2. Each entry of Table IV contains only three numbers which are in the order of $NT_{i/j,d}.dly/NT'_{i/j,d}.dly/NT''_{i/j,d}.dly$ because all "$.num$" entries are either 0 or 1. A $*$ symbol marks the entry where the optimal path is found for the first time after the link failure, and an ∞ symbol indicates paths with an infinite delay or paths which include a loop. Some other entries of NT' needed in our algorithm are decribed in Table V.

TABLE IV
DELAY TABLES OF ALL NODES IN S_1 WITH THE DESTINATION NODE N_3 UNDER OUR ALGORITHM

	Time						
entry	0	1	2	3	4	5	6
Delay Table of N_1 to destination N_3							
N_2	5/3/∞	∞/∞/3	∞/∞/3	∞/∞/3	∞/∞/3	142/140/3	142/140/3
N_{11}	74/41/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	141/108/∞	141/108/∞	141/108/∞
N_{12}	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	118*/109/∞	118/109/∞
Delay Table of N_2 to destination N_3							
N_1	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	143/141/∞	120*/118/∞
N_3	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞
N_{13}	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	140/135/∞	140/135/∞	140/135/∞
Delay Table of N_9 to destination N_3							
N_8	105/5/∞	105/5/∞	105/5/∞	105*/5/∞	105/5/∞	105/5/∞	105/5/∞
N_{10}	81/21/∞	81/21/∞	81/21/∞	∞/∞/∞	107/47/∞	∞/∞/47	∞/∞/47
N_{11}	18/15/∞	18/15/∞	18/15/∞	∞/∞/15	∞/∞/15	∞/∞/15	∞/∞/15
N_{13}	38/8/∞	38/8/∞	∞/∞/8	∞/∞/8	∞/∞/8	∞/∞/8	∞/∞/8
Delay Table of N_{10} to destination N_3							
N_2	78/18/∞	78/18/∞	78/18/∞	∞/∞/∞	165/105/∞	165/105/∞	165/105/∞
N_{11}	21/15/∞	21/15/∞	21/15/∞	47/41/15	114*/108/15	114/108/15	114/108/15
Delay Table of N_{11} to destination N_3							
N_1	38/5/∞	38/5/∞	∞/∞/5	∞/∞/5	∞/∞/5	∞/∞/5	175/142/5
N_9	41/38/∞	41/38/∞	41/38/∞	108*/105/38	108/105/38	108/105/38	108/105/38
N_{10}	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	171/165/∞
N_{12}	15/14/∞	15/14/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞	∞/∞/∞
Delay Table of N_{12} to destination N_3							
N_1	14/5/∞	14/5/∞	∞/∞/5	∞/∞/5	∞/∞/5	150/141/5	150/141/5
N_{11}	39/38/∞	39/38/∞	∞/∞/∞	42/41/∞	109*/108/41	109/108/41	109/108/41

	Time					
	0	1	2,3	4,5	6	7
Delay Table of N_{13} to destination N_3						
N_2	8/3/∞	∞/∞/3	∞/∞/3	∞/∞/3	148/143/3	125*/120/3
N_9	48/18/∞	∞/∞/∞	∞/∞/∞	135/105/∞	135/105/∞	135/105/∞

TABLE V
USEFUL ENTRIES

	$NT'_{1/11,2}$	$NT'_{1/12,2}$	$NT'_{9/10,11}$	$NT'_{10/9,11}$	$NT'_{11/12,1}$	$NT'_{12/11,1}$	$NT'_{13/9,2}$
delay	38	39	6	3	9	33	15

V. SIMULATION

We simulate both APRS and the proposed strategy to compare their performance. Two networks (in Figs. 8 and 9) are used in the simulation. Packets are assumed to arrive at the system according to a Poisson process. During the simulation, we increase the rate of packet arrival until the network under testing gets overloaded. The packet size is assumed to be uniformly distributed between 20 and 1000 bytes. The period of exchanging routing tables between adjacent nodes is assumed to be 128 ms (as in APRS). The number attached to each link in these figures represents the delay (in units of 0.1 ms) of transmitting a 1000-byte packet over that link; for example, a 100 kb/s link has label 100 and a 10 kb/s link has label 1000.

The simulation has two parts: without and with component failures. In Figs. 10 and 11, the vertical axis represents the ratio of packet delivery delay of the proposed strategy to that of APRS. We used the average of the data collected in a period of 100 s as a sample; for example, under 200 packets/s load, each sample represents the average of approximately 20 000 packets. Figs. 10 and 11 show the average value of 100 samples under specific load conditions. In both figures, most samples we collected were around the average value. However, there were a few samples (around 1%) with large variances. When the network is lightly loaded, the two strategies exhibit similar values of packet delivery delay. However, when the network load increases, the APRS delivery delay increases much faster than our strategy. Moreover, since the delay increases faster, the packets start to build up at local hosts under APRS when the packet arrival rates are greater than 400 packets/s in Fig. 10 and 500 packets/s in Fig. 11. By contrast, our strategy can still function correctly at those load levels. As the delay ratio in the 20-node network drops much faster than it does in the ten-node network, our strategy is expected to perform even better for larger networks. When the network becomes larger, the looping effect caused by the link delay changes (due to increasing queueing delay) has more pronounced effects on the average packet delivery delay since the loop is likely to be larger in a larger network, and APRS likely needs a longer time to resolve a larger loop.

In the second part of the simulation, assuming the occurrence of a link failure, we measured the average delay per packet for the first 100 s after the failure. As in the first part of simulation, each point in Figs. 12 and 13 is the average

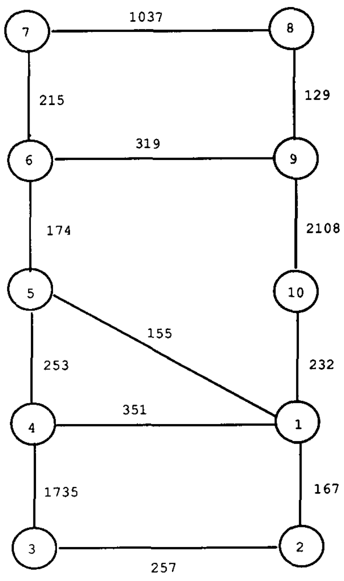


Fig. 8. The ten-node test network.

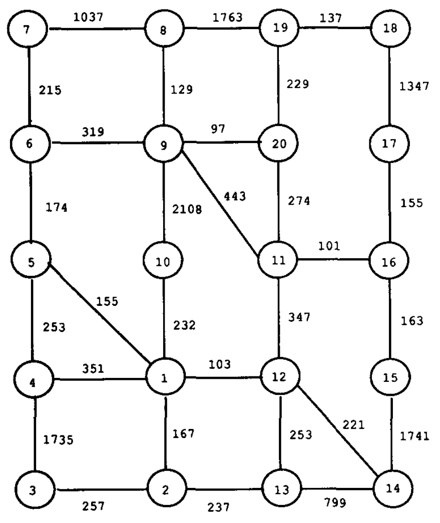


Fig. 9. The 20-node test network.

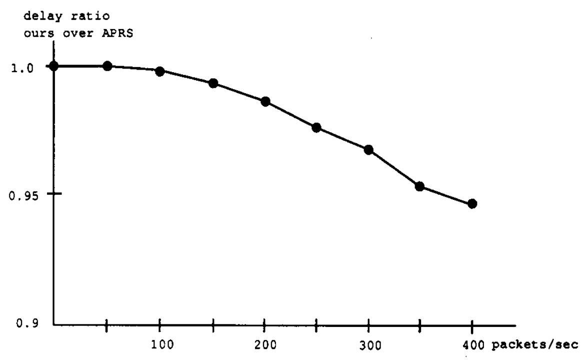


Fig. 10. Ratio of average per-packet delays in the ten-node network.

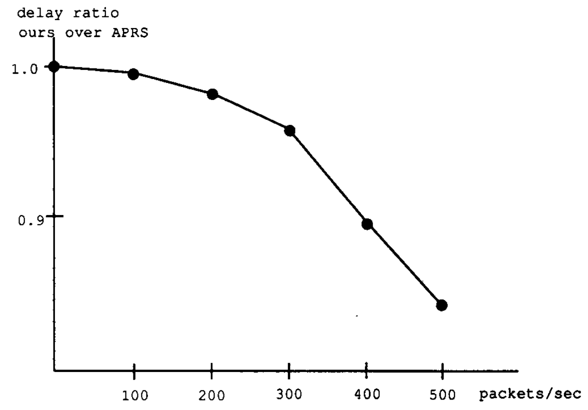


Fig. 11. Ratio of average per-packet delays in the 20-node network.

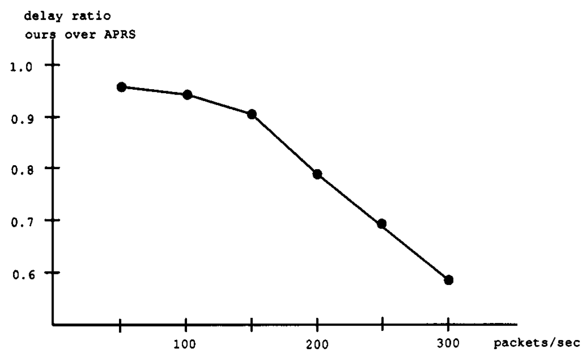


Fig. 12. Ratio of average delays with the presence of failure in the ten-node network.

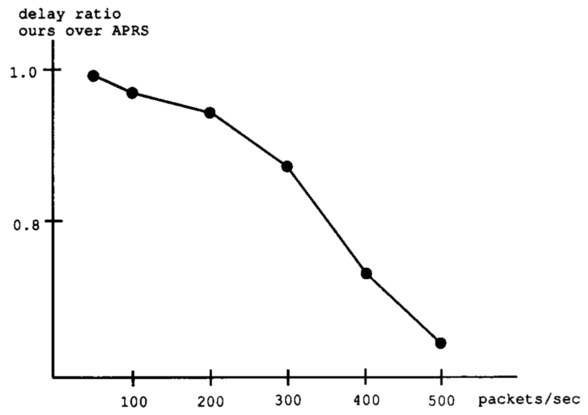


Fig. 13. Ratio of average delays with the presence of failure in the 20-node network.

of 100 samples which are mostly around the average value. However, a few of them (around 1%) are somewhat off from the average value. The vertical axis again represents the ratio of packet delivery delay of our strategy to that of APRS. In the presence of a failure, our strategy outperforms APRS because of its loop-free capability. The tendencies of Figs. 12 and 13 are the same as in Figs. 10 and 11, except that the ratio figure drops much faster. Due to the presence of component failures, the looping effect of APRS is more pronounced than it was in the first part. Since the failed link has an infinite delay, all paths going through that link suffer the looping problem. By contrast, our strategy is loop-free, and therefore, the ratio drops much faster in the presence of component failures. Because of a larger average packet delivery delay the networks under APRS become overloaded when the arrival rate reaches 300 (in the ten-node network) and 500 (in the 20-node network) packets/s.

VI. Conclusion

In this paper, we propose a very simple but effective loop-free routing strategy which can completely solve the looping problem in case of link-delay changes as well as link/node failures. We have also proved the correctness of the strategy and analyzed its performance and operational overheads. The performance is improved significantly by simply attaching the total number of paths with the optimal delay between two nodes to the normal routing messages under APRS. Moreover, the operational overhead of the proposed algorithm is shown to be very low. Both ours and APRS have time and space complexity $O(n)$ for sparse networks, where n is the number of nodes in the network. Despite its simplicity, the proposed algorithm can eliminate the looping problem completely with little overhead.

List of Symbols

N_i	Node i.
$L_{i,j}$	The directional link from N_i to N_j.
$DL_{i,j}(m)$	The delay of link $L_{i,j}$ at time m.
$DL_{i,j}$	The delay of link $L_{i,j}$ when time is immaterial.
$OP_{s,d}(m)$	The optimal (least delay) path from N_s to N_d at time m.
$DOP_{s,d}(m)$	The delay of $OP_{s,d}(m)$.
A_i	The set of nodes which are adjacent to N_i.
$D_{i/j,d}(m)$	The delay of the path from N_i through $N_j \in A_i$ to N_d in N_i's table.
$d_{i,j}$	The delay of the optimal paths from N_i to N_j when time is immaterial.
$n_{i,j}$	Number of the optimal paths from N_i to N_j when time is immaterial.

$SP^t_{jd\backslash i}$	Set of optimal paths from $N_j \in A_i$ to N_d without passing through N_i at time t.
$P^t_{jd\backslash i}$	Representative of $SP^t_{jd\backslash i}$.
$n^t_{jd\backslash i}$	Number of elements (paths) in set $SP^t_{jd\backslash i}$.
$D(P)$	Delay function of a path P.
$RM_{i\leftarrow j,d}$	Routing messages which are sent from N_j to $N_i \in A_j$ for the destination N_d.
$RM_{i\leftarrow j,d}.dly$	The delay field of $RM_{i\leftarrow j,d}$.
$RM_{i\leftarrow j,d}.num$	The number field of $RM_{i\leftarrow j,d}$.
$NT'_{i/j,d}$	The entry containing the optimal paths from N_j to N_d without passing N_i in N_i's table.
$NT'_{i/j,d}.dly$	The delay field of $NT'_{i/j,d}$.
$NT'_{i/j,d}.num$	The number field of $NT'_{i/j,d}$.
$NT''_{i/j,d}$	The entry containing the known faulty paths from N_j to N_d without passing N_i in N_i's table.
$NT''_{i/j,d}.dly$	The delay field of $NT''_{i/j,d}$.
$NT''_{i/j,d}.num$	The number field of $NT''_{i/j,d}$.
$NT_{i/j,d}$	The entry containing the delay of the optimal paths from N_i through $N_j \in A_i$ to N_d in N_i's table.
$scn(P)$	The screen function of a path P.
$h(P)$	The hop function of a path P.

REFERENCES

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

[2] B. W. Boehm and R. L. Mobley, "Adaptive routing techniques for distributed communications systems," *IEEE Trans. Commun. Technol.,* vol. COM-17, pp. 340–349, June 1969.

[3] T. Cegrell, "A routing procedure for the TIDAS message-switching network," *IEEE Trans. Commun.,* vol. COM-23, pp. 575–585, June 1975.

[4] C. C. Cheng, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves, "A distributed loop-free rooting algorithm suitable for arbitrary link weights," Tech. Rep. CSS-89-05, Dep. Elec. Eng. Comput. Sci., Northwestern Univ., Sept. 1989.

[5] J. M. Jaffe and F. H. Moss, "A responsive distributed routing algorithm for computer networks," *IEEE Trans. Commun.,* vol. COM-30, pp. 1758–1762, July 1982.

[6] M. J. Johnson, "Updating routing tables after resource failure in a distributed computer network," *Networks,* vol. 14, pp. 379–391, 1984.

[7] L. Kleinrock and H. Opderbeck, "Throughput in the ARPANET-protocols and measurement," *IEEE Trans. Commun.,* vol. COM-25, pp. 95–103, Jan. 1977.

[8] J. M. McQuillan, I. Richer, and E. C. Rosen, "The new routing algorithm for the ARPANET," *IEEE Trans. Commun.,* vol. COM-28, pp. 711–719, May 1980.

[9] P. M. Merlin and A. Segall, "A failsafe distributed routing protocol," *IEEE Trans. Commun.,* vol. COM-27, pp. 1280–1288, Sept. 1979.

[10] D. J. Nelson, K. Sayood, and H. Chang, "An extended least-hop distributed routing algorithm," *IEEE Trans. Commun.,* vol. 38, pp. 520–528, Apr. 1990.

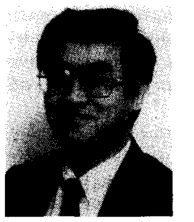
[11] K. Ramamritham, J. A. Stankovic, and W. Zhou, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Trans. Comput.,* vol. 38, pp. 1110–1122, Aug. 1989.

[12] K. G. Shin and M.-S. Chen, "Performance analysis of distributed routing strategies free of ping-pong-type looping," *IEEE Trans. Comput.,* vol. C-36, pp. 129–137, Feb. 1987.

[13] ——, "Minimal order loop-free routing strategy," *IEEE Trans. Comput.,* vol. 39, pp. 870–888, July 1990.

[14] W. D. Tajibnapis, "A correctness proof of a topology information maintenance protocol for distributed computer networks," *Commun. Ass. Comput. Mach.,* vol. 20, pp. 477–485, 1977.

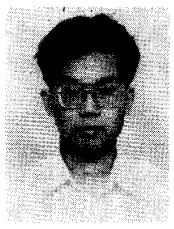
[15] A. S. Tanenbaum, *Computer Networks,* 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1989.



Kang G. Shin (S'75–M'78–SM'83–F'92) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and both the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY, in 1976 and 1978, respectively.

He is Professor and Chair of the Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, NY. He has held visiting positions at the U.S. Air Force Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at UC Berkeley, and International Computer Science Institute, Berkeley, CA.

Dr. Shin 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, and is a Program Co-Chair for the 1992 INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING. He chaired the IEEE Technical Committee on Real-Time Systems from 1991 to 1993, served as a Distinguished Visitor of the Computer Society of the IEEE, is an Editor of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED COMPUTING, and is an Area Editor of *International Journal of Time-Critical Computing Systems.* He has authored/coauthored over 240 technical papers (about 110 of these in archival journals) and several book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, and robotics and automation. In 1987, he received the Outstanding IEEE T RANSACTIONS ON AUTOMATIC CONTROL Paper Award for a paper on robot trajectory planning. In 1989, he also 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 currently building a 19-node hexagonal mesh multicomputer, called HARTS, to validate various architectures and analytic results in the area of distributed real-time computing.



Chih-Che Chou received the B.S.E. degree from National Taiwan University, Taipei, Taiwan, in 1988, and the M.S.E. degree from the University of Michigan, Ann Arbor, in 1992.

Currently, he is working toward the Ph.D. degree in computer science and engineering at the University of Michigan. His research interests include real-time communication, distributed systems, and communication systems for manufacturing.