Design and Evaluation of Routing Schemes for Dependable Real-Time Connections*

Songkuk Kim, Daji Qiao, Sharath Kodase, and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, MI 48109-2122
Phones: 734-763-0391 (voice); 734-763-8094 (fax)
E-mail: {songkuk,dqiao,skodase,kgshin}@eecs.umich.edu

Abstract

Dependability-of-Service (DoS) has become an important requirement for real-time applications, such as remote medical services, business-critical network meetings, and command & control applications. The Dependable Real-Time Protocol (DRTP) [5, 6, 7] in which each dependable real-time connection is realized with one primary and one or more backup channels, has been shown to be an effective way of providing DoS. How to route both primary and backup channels for each dependable real-time connection is of vital importance to the success of failure recovery and the reduction of overhead in providing DoS.

In this paper, we propose and evaluate three different schemes for routing the primary and backup channels of each dependable real-time connection. Specifically, we present methods based on link-state information and bounded flooding to discover routes for primary and backup channels while satisfying the required Quality-of-Service (QoS). The costs of link-state and flooding algorithms are reduced significantly by using the fact that the probability of success in failure recovery can be estimated with simple link-state information, and by bounding the flooded region within an ellipse with the two communication end-points as loci. Our in-depth simulations have shown that the proposed routing schemes are highly effective, providing 80% or higher of fault-tolerance with overhead al most 25% of the case without any backups.

*The work reported in this paper was supported in part by the Office of Naval Research under Grant No. N00014-99-1-0299. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the ONR.

Keywords: Dependable real-time (DR-) connection, dependable real-time protocol (DRTP), primary and backup channels, backup multiplexing, link-state routing, bounded flooding

1. Introduction

Real-time transport of continuous media has traditionally been achieved by circuit switching in telephony services or by broadcasting over shared media in television services. In packet-switched networks, however, continuous media requires a special care since the end-to-end packet delay and throughput of a media stream are inherently non-deterministic. Such end-to-end performance characteristics which are necessary to achieve the required functionality of these applications are often called *Quality-of-Service* (QoS). Typical performance QoS includes message throughput, end-to-end delay and delay jitter.

In recent years, the rapid improvement of network connectivity and link capacity has expanded the application domain of real-time communication service to safety- and business- critical applications, such as remote medical services, business video conferences, and military command, control & communication. These applications require support for *Dependability-of-Service* (DoS) — in addition to support for performance QoS — in order to deal with network failures which are more likely to occur as the network gets larger and more complex. For DoS support, one must consider both transient and persistent network failures. A typical example of transient failures is temporary packet losses due to either network congestion or data corruption. Persistent failures include breakdown of network components (links and switches). Reliable transport protocols like TCP can handle transient packet losses by acknowledgment and retransmission, and forward-error-correction (FEC) can also be used to deal with transient failures, particularly for real-time communication service. To handle persistent network failures, various dependability schemes have been proposed, which can be broadly classified as *reactive* or *proactive*.

Reactive schemes deal with failures only *after* their occurrences [3]. To restore a real-time connection from a network component failure, one has to set up a new real-time connection which does not include any faulty components. Since no resource is reserved *a priori* for the purpose of fault-tolerance, this method does not incur overhead in the absence of failures. However, it cannot give any guarantee on failure recovery due to potential resource shortage and/or contention in attempting recovery from failures. Banerjea extended this approach further in [2] by proposing *delayed* retries to spread simultaneous recovery attempts. He suggested a random delay before starting each recovery process and a retry along the same path with an exponential back-off in case the recovery process fails. However, this method may require several trials to succeed, thus delaying service resumption and increasing network traffic. The recovery can take several seconds or longer, especially in heavily-loaded networks.

Proactive schemes achieve dependability by means of additional resources reserved *a priori* in the network. In the multi-copy method [8, 10], multiple copies of a packet are sent simultaneously via disjoint paths. This method attempts to achieve both timely and reliable delivery at the same time. Although it can handle network failures without service disruption, this method introduces a large resource overhead and cannot guarantee timely delivery due to its reliance on best-effort delivery of packet copies. Dispersity routing [1] combines forward-error-correction with multiple-copy transmissions, which allows for a tradeoff between resource overhead and fault-tolerance capability. In the Single Failure Immune (SFI) method [12], additional resources are reserved in the vicinity of each real-time channel, and the failed components are detoured by using the reserved resources. In [13], the SFI method was extended to combat special patterns of multiple failures in a hexagonal mesh network.

Spare resource allocation [4, 5, 6, 7] is another approach that employs failure detection and recovery for fault-tolerance. Additional (called *spare*) resources are reserved *a priori* solely for the purpose of speedy service recovery from possible network failures. The key advantage of proactive schemes is that the latency and success-probability of service recovery are usually better than those of the reactive schemes. Note, however, that the spare resources cannot be used to accommodate other real-time connections, although they might be used for transporting best-effort traffic. Therefore, given the same amount of network resources, the proactive schemes usually result in a lower network utilization than the reactive schemes. Two main issues in developing proactive schemes are to reduce and bound the service-recovery latency and to minimize the fault-tolerance overhead.

Han and Shin [5, 6, 7] proposed the *dependable real-time protocol* (DRTP), a typical spare resource allocation scheme, which consists of the following four steps: (1) establishment of primary and backup channels, (2) detection of network failures, (3) failure reporting and channel switching, and (4) resource reconfiguration. How to route the primary and backup channels for a dependable real-time connection is a key element of DRTP, which, despite its importance, has not yet been addressed adequately. In this paper, we propose three different routing schemes for primary and backup channels, and comparatively evaluate their performances in terms of fault-tolerance and resource overhead.

2. Dependable Real-Time Protocol

Each dependable real-time (DR-) connection consists of one *primary* and one or more *backup* channels. Upon detection of a failure on the primary channel, one of its backups is *promoted* to the new primary. Since a backup is set up before a failure of the primary, it can be activated immediately, without the time-consuming, and sometimes unsuccessful, channel (re)-establishment process.

A backup channel does not carry any real-time traffic[1] until it is activated, and hence, it does not consume resources in the absence of failures. However, a backup channel is not free, since it requires reservation of at least[2] as much resources as its primary channel. As a result, equipping each DR-connection even with a single backup disjoint from its primary reduces the network capacity by at least 50%, which is too expensive to be practically useful. To deal with this problem, a resource-sharing technique, called *backup multiplexing*, was introduced in DRTP [5, 6, 7]. The basic idea of backup multiplexing is that, on each link, instead of reserving all of the resources necessary for each backup, only a small fraction of the necessary resources is reserved and then shared by all backups running through

[1] It may carry best-effort traffic, though.

[2] Note that a backup may run through a longer path than the corresponding primary.

the link, i.e., *overbooking* link resources for backups. The amount of total necessary spare resources is determined on a hop-by-hop basis by considering the relation among all the backups traversing the same link.

The fault-tolerance of a DR-connection depends primarily on the probability of backup activation. The backup activation can fail when the spare resources are multiplexed on backups to reduce the resource overhead. Resource overbooking and sharing by backups would be acceptable if their corresponding primaries are disjoint.

Backup channels are said to have *conflicts* if they traverse the same link and their corresponding primaries overlap, or share link(s). Some of the conflicting backups multiplexed over the same spare resources may fail to be activated when their corresponding primaries fail (near-)simultaneously. To provide better fault-tolerance, backup conflicts should be minimized, and in the case of a conflict, backup multiplexing should be avoided or minimized.

Figure 1 illustrates the idea of backup multiplexing using a simple 3×3 mesh network. Each connection between two nodes has two unidirectional links. Although there are 24 uni-directional links, we only consider 13 of them in the following examples. There are three DR-connections D_1, D_2, and D_3. The primary and backup channels of these connections are shown with solid and dashed arrows, respectively. In this example, we assume that only a single link can fail between two successive recovery actions. Consider link L_6, which is part of the routes of the backup channels B_1 and B_2. Because the primary channel P_1 and P_2 do not overlap, any single link failure can cause at most one of these primaries to be switched to its backup. Thus, B_1 and B_2 will never contend for the reserved resources on L_6, and therefore, the backup multiplexing on L_6 successfully reduces the resource overhead without affecting the fault-tolerance capability. Now, let's consider link L_7, which is used by the backup routes B_1 and B_3. Since P_1 and P_3 overlap at L_{13}, if L_{13} fails, both DR-connections need to be switched to their backups. Hence, the resource needs on L_7 exceed the reserved amount, and L_7 can accommodate only one connection. As a result, one of the DR-connections will fail to activate its backup. If D_3's QoS requirement (e.g., end-to-end delay) is too tight to use the longer path, then it cannot be recovered from the failure of L_{13}. Therefore, the backup multiplexing on L_7 degrades the fault-tolerance capability.

The above observations show that routing channels under backup multiplexing has a significant bearing on the resulting fault-tolerance capability. An ideal backup channel B should (1) provide the same QoS as its primary upon its activation; (2) overlap minimally with its primary; and (3) overlap minimally with other backups whose primaries overlap with B's primary.

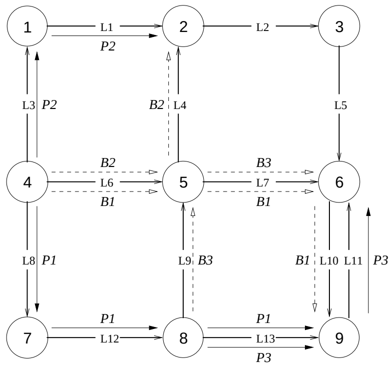


Figure 1. An example of backup multiplexing

To find a backup route that meets these three requirements, one must know where primary channel paths run, where the corresponding backup paths are, and the amount of resources available on these paths. However, it is not scalable to require every router to keep all this information. Especially, maintaining information on all DR-connections at each router is impractical because the required amount of information is $O(n \times average-path-length)$, where n is the number of DR-connections. Thus, we develop a mechanism that requires every router to maintain only *abridged* information. We will revisit this in Section 3. In this paper, we propose three routing schemes for DR-connections.

2.1. Notation

We use the following symbols/notation.

- N: the total number of links in the network.
- E: average node degree of the network.
- P_i: the primary channel of DR-connection D_i.
- B_i: the backup channel of DR-connection D_i.
- $total\_bw$: total bandwidth can be used for DR-connections.
- $prime\_bw$: bandwidth consumed by the primary channels.
- $spare\_bw$: bandwidth reserved by the backup channels.
- $LSET_r$: the set of links in route r.
- $PSET_i$: the set of primary routes whose backup routes go through link L_i.
- $APLV_i$: Accumulated Primary route Link Vector whose j^{th} element, denoted by $a_{i,j}$, represents the total number of primary channels that traverse link L_j and whose backup channels go through link L_i. Then, $a_{i,j} = |\{P_k : P_k \in PSET_i$ and $L_j \in LSET_{P_k}\}|$,

- $\|APLV_i\|_1$: the L_1-norm of $ARLV_i$, which is defined as $\sum_{j=1}^{N} a_{i,j}$.
- SC_i: the number of backups on L_i that can be activated successfully using the spare resources.

2.2. DR-Connection Management

To support the DR-connection service, every router is equipped with a DR-connection manager which consists of two modules; one routes backup channels and the other multiplexes backups. The former exchanges and maintains the information necessary to select backup routes. We assume that a portion of network resources is set aside for DR-connections. The total amount of resources for DR-connections cannot exceed this portion, and these resources can be used for non-real-time traffic when they are not used by DR-connections.

Management of each DR-connection consists of the following four steps.

1. Select a primary route and reserve resources when a client/server node requests a DR-connection to be set up.
2. Find a backup route after establishing the primary.
3. Send a *backup-path register* packet along the newly-selected path.
4. Release the resources of the primary and backup routes when the DR-connection is terminated.

Every router maintains $APLV$ for each of links attached to it, but the entire $APLV$s are not stored in each router's link-state database. $APLV$ is used for routing and multiplexing backups. To maintain $APLV$ for a link, a router needs $LSET$s of its $PSET$. Storing all $LSET$s may require large memory space, because an excessive number of backup channels can go through a single high-speed link. To cope with this problem, when a node sets up or releases a backup channel, it sends the $LSET$ of the corresponding primary route in a *backup-setup request*. When a router receives a backup-setup request, it checks the amount of available resources. The router registers this new backup in the backup channel table and updates $APLV$ for the link that the backup channel traverses using $LSET$. Finally, the router forwards the request to the next router in the backup path. When a router rejects the request for setting up a backup channel, it sends a *backup-release request* in which $LSET$ of the corresponding primary route is included.

3. Link-State Routing Schemes

A node can select a backup path that has minimum conflicts, if it has complete knowledge of $APLV$s for all the links in the network. The j^{th} element of $APLV_i$ represents the number of DR-connections whose backups and primaries traverse L_i and L_j, respectively. In Figure 1, we have $PSET_7 = \{P_1, P_3\}$, $LSET_{P_1} = \{L_8, L_{12}, L_{13}\}$, $LSET_{P_2} = \{L_1, L_3\}$, and $APLV_7 = (0,0,0,0,0,0,0,1,0,0,1,1,2)$. $APLV_7$ indicates that if L_7 is selected as a link of the backup route for a DR-connection whose primary channel goes through L_{12}, it will generate conflicts with two other backups. $APLV_i$ represents the number and the positions of backup conflicts that will occur when L_i is used as a backup path link. Thus, if a node has knowledge of all $APLV$s in the network, it can select a backup path that will create minimum conflicts. However, it is too costly for every node to acquire and maintain all $APLV$s, as there are N APLVs, each with N integers.

We therefore develop two link-state routing schemes that use an abridged form of $APLV$: (1) P-LSR that infers and exploits the probability of backup conflicts using $\|APLV\|_1$ and (2) D-LSR that uses a bit-vector form of $APLV$, called *Conflicts Vector*, to decide deterministically if a link has backup conflicts.

3.1. P-LSR: Probabilistic Avoidance of Backup Conflicts

The idea behind P-LSR is that the probability of link L_i's backup conflicts rises as the number of links in $PSET_i$, which is equal to $\|APLV_i\|_1$, increases. Without knowing where primary routes run, it is very logical to select a link with a smaller $\|APLV\|_1$ to minimize backup conflicts and maximize the probability of successful backup activation on L_i. In Figure 1, $\|APLV_2\|_1 = 0$, $\|APLV_4\|_1 = 2$, and $\|APLV_7\|_1 = 5$. As a link for a backup route, L_2 is preferable to L_4 or L_7.

To activate a backup, B_x, on L_i without any conflict, P_k should be disjoint from all the primary routes whose backups traverse L_i, i.e., $LSET_{P_x} \cap \{\bigcup_{P_j \in PSET_i} LSET_{P_j}\} = \emptyset$. Let $\phi_{B_x,i}$ denote the probability of successful activation of B_k on L_i. Then, $\phi_{B_x,i}$ can be calculated as:

$$\phi_{B_x,i} = \left(\frac{N - |LSET_{P_x}|}{N}\right)^{\|APLV_i\|_1}. \qquad (1)$$

Since our goal is to select a backup route that has the maximum probability of successfully activation, we need to know the relation between the probability of backup activation and links' $\|APLV\|$ in the backup route. The relation can be derived easily as follows. Consider a DR-connection D_x whose primary channel P_x has already been established. Our goal is then to find the best backup route, B_x, by maximizing the probability of successful activation upon P_x's failure.

The probability of successfully activating B_x, denoted by Φ_{B_x}, can be calculated by

$$\Phi_{B_x} = \prod_{L_i \in LSET_{B_x}} \phi_{B_x,i}. \tag{2}$$

Since the log function is monotonic, maximizing Φ_{B_x} is equivalent to maximizing $\log \Phi_{B_x}$ where

$$\begin{aligned} \log \Phi_{B_x} &= \log \left(\prod_{L_i \in LSET_{B_x}} \phi_{B_x,i} \right) && (3) \\ &= \sum_{L_i \in LSET_{B_x}} \log \left(M^{\|APLV_i\|_1} \right) \\ &= (\log M) \times \left(\sum_{L_i \in LSET_{B_x}} \|APLV_i\|_1 \right) && (4) \end{aligned}$$

where $M = \frac{N - |LSET_{P_x}|}{N}$. Since $\log M$ is a negative constant, a path B_x that has minimum $\sum_{\ell \in LSET_{B_x}} \|APLV_\ell\|_1$ will maximize the probability of backup activation. Such a path can be found using the Dijkstra's algorithm by assigning $\|APLV_i\|_1$ as the cost for link L_i.

$\|APLV\|_1$ and the available bandwidth (the sum of the un-allocated bandwidth and the spare bandwidth shared by the backup channels) are stored in each link-state database. To select a backup path after establishing a primary channel, a router assigns C_i as the cost of link L_i, and chooses a minimum-cost path using the Dijkstra's algorithm, where

$$C_i = Q + \|APLV_i\|_1 + \varepsilon. \tag{5}$$

Q is a very large constant ($\gg max(APLV_i)$) if P_x traverses L_i or the available bandwidth is smaller than the QoS requirement, 0 otherwise. A small positive constant, ε ($\ll 1$), is used to select the shortest route as the backup path if there are several candidate routes with the same degree of channel overlapping. The resulting path will be the shortest backup route that meets the QoS requirement, minimally overlaps with its corresponding primary channel route, and maximizes the probability of successful activation.

3.2. D-LSR: Deterministic Avoidance of Backup Conflicts Using Conflict-Vector

$APLV$ contains information on the number and position of backup conflicts. D-LSR uses a simple data structure, *Conflict-Vector* (CV) that shows only the location of backup conflicts. The CV of link L_i, denoted by CV_i, is an N-element bit-vector, the j^{th} element of which, $c_{i,j}$, is 1 if the j^{th} element of $APLV_i$, $a_{i,j}, > 0$; 0 otherwise. Thus, $c_{i,j} = 1$ if and only if there is at least one primary channel running through L_j whose backup traverses L_i.

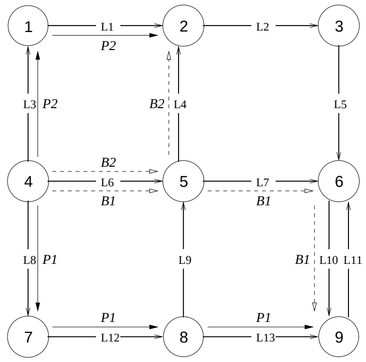


Figure 2. An example network topology

Figure 2 shows a simple exaple with two DR-connections D_1 and D_2. Since both backup channels, B_1 and B_2, go through L_6, $PSET_6 = \{P_1, P_2\}$. From $LSET_{P_1}$ and $LSET_{P_2}$, one can easily compute $APLV_6$ and $CV_6 = (1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1)$. The CVs for the other links can be computed similarly.

After establishing the primary P_x, the node can use CV_i and $LSET_{P_x}$ to check if L_i creates backup conflicts to choose a link for backup. If $L_j \in LSET_{P_x}$ and $c_{i,j} = 1$, L_i will introduce backup conflicts. Thus, the node selects L_i such that $\sum_{L_j \in LSET_{P_x}} c_{i,j}$ is minimum. To choose a backup route with minimum conflicts while meeting the QoS requirement, the node assigns L_i the link cost C_i and selects the minimum-cost route using the Dijkstra's algorithm, where

$$C_i = Q + \sum_{L_j \in LSET_{P_x}} c_{i,j} + \varepsilon. \tag{6}$$

Q and ε are added for the same reason as in P-LSR. In D-LSR, CVs and the available bandwidths are stored in the link-state database.

Consider the example network in Figure 3, where two DR-connections are established and node 8 selects the backup for D_3 whose primary is running through L_{13} and L_{11}. Suppose the links have enough bandwidths available to provide the required QoS. (L_9, L_4, L_2, L_5) is selected as the backup channel route, B_3^*, of the DR-connection. Note that if L_{13} fails, both connections a and c fail simultaneously. However, since the backup routes are disjoint, both connections can recover from the link failure. As compared to B_3 in Figure 1, B_3^* offers better fault-tolerance than B_3, although it has a longer distance.

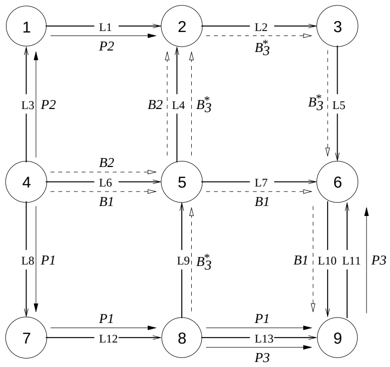


Figure 3. Backup route selection for D_3

4. Routing with Bounded Flooding

Link-state routing is easy to implement, but the extended link-state packet requires a larger packet size and introduces additional routing traffic. To deal with this problem, we propose a different on-demand routing scheme based on bounded flooding, which was originally proposed by Kweon and Shin [9] for QoS (not DoS) routing.

Suppose a destination (client) node requests a DR-connection service from the source (server) node, and specifies its QoS requirement by indicating the minimum desired link bandwidth of the connection. In order to establish the DR-connection, the source node floods a special channel-discovery packet (CDP) towards the destination. To reduce the overhead, the source node limits the number of hops each CDP can take before reaching the destination. That is, when an intermediate node receives a CDP, it will forward the packet to its neighbors only if the minimum-hop route via that neighbor can lead the CDP to the destination within the source-specified hop-count limit. This scheme can be viewed as *bounded flooding*. The destination node is responsible for selecting the best primary and backup routes for the real-time connection based on the flooded information. Before proceeding to the proposed algorithm, we introduce the relevant data structures.

4.1. Data Structures and Notation

Each network node maintains a distance table (DT). Let NB_i denote the set of node i's neighbors. Hop count is used to build distance tables, although any other distance metric can be used. The distance table at node i is a 2-dimensional matrix containing, for each destination j and for each neighbor $k \in NB_i$, the minimum hop count from i to j via k, which is denoted by $D^i_{j,k}$. So, the minimum distance from node i to destination j is

$$D^i_j = \min_{k \in NB_i} D^i_{j,k} + 1. \tag{7}$$

The minimum hop count can be calculated using the Dijkstra's algorithm or the Bellman-Ford distance-vector algorithm. The distance tables are updated only upon change of the network topology.

To establish a DR-connection from the source i to the destination j, the source initiates the bounded flooding of a CDP, which contains the following fields:

- *srce_id* (*dest_id*): an integer which uniquely identifies the source (destination) node.
- *conn_id*: a number that uniquely identifies a DR-connection.
- *hc_limit*: maximum hop count that the CDP can take before reaching the destination.
- *hc_curr*: hop count of the route taken so far by the CDP to reach the current node.
- *bw_req*: bandwidth requested for the DR-connection.
- *primary_flag*: one bit flag to indicate if the route traversed so far by the CDP can be used for primary route. It is 1 if *total_bw* − (*prime_bw* + *spare_bw*) is larger than the *bw_req*, otherwise 0.
- *list* of nodes that the CDP has traversed so far. Every time the CDP is forwarded, the current node is appended to *list*. This information is needed for the destination to select the best routes for the primary and backup channels of the connection, and to guarantee loop-free flooding.

The flooding bound of the CDP is specified by *hc_limit*, which is equal to $\rho \times D^i_j + \mu$, where $\rho \geq 1$ and $\mu \geq 0$. In order to improve the chance of granting the requested DR-connection, multiple routes must be given opportunities to run the connection over them. Therefore, the values of ρ and μ are determined by making a trade-off between the routing overhead and the connection-acceptance probability.

Each node maintains a "transient" table, Each entry of a PCT represents a connection request passed through this node, and consists of four fields:

- *conn_id*: the connection identifier.
- *bw_req*: the requested bandwidth.
- *min_dist*: hop count of the shortest route taken by CDPs to the current node.
- *time_out*: a real number which specifies this entry's time to live. Upon expiration of the timer, this entry is no longer valid and thus deleted from the PCT. In order to prevent a false deletion, *time_out* must be no less than the average link delay multiplied by the hop-count limit.

Besides PCT, each node maintains a set of candidate route tables (CRTs), one for each outstanding connection request destined for this node. The function of these tables is to allow the destination to choose the best primary and backup routes among those routes which the CDPs have safely traversed to reach the destination. Each entry of a CRT represents one candidate route for the corresponding connection request, and contains $primary\_flag$, hop_count and $list$.

4.2. Action by the Source Node

The destination node initiates a connection request and uni-casts the request message to the source node. Upon receiving the request message, the source node i composes a CDP m, and performs the following tests for each of its neighbors $k \in NB_i$:

Distance test:

$$D^i_{dest\_id,k} + 1 \leq hc\_limit(m). \tag{8}$$

Bandwidth test:

$$bw\_req(m) \leq total\_bw(i,k) - prime\_bw(i,k). \tag{9}$$

If node k passes both tests, node i updates and forwards the packet to node k. The CDP is updated by recalculating $primary\_flag(m)$, increasing $hc\_curr(m)$ by one, and appending i to $list(m)$.

4.3. Action by an Intermediate Node

Upon receiving a CDP, m, node i performs the following tests for each of its neighbors $k \in NB_i$:

Distance test:

$$hc\_curr(m) + D^i_{dest\_id,k} + 1 \leq hc\_limit(m). \tag{10}$$

Loop-freedom test:

$$k \notin list(m). \tag{11}$$

Bandwidth test:

$$bw\_req(m) \leq total\_bw(i,k) - prime\_bw(i,k). \tag{12}$$

If node i has already received at least one copy of the CDP for the same DR-connection, node i performs an additional test on the incoming CDP before executing the above three tests:

Valid-detour test:

$$hc\_curr(m) \leq \alpha \times min\_dist(conn\_id(m)) + \beta. \tag{13}$$

By using this additional "valid-detour test," where α and β are two pre-determined parameters, we further reduce the number of CDPs. If node k passes all these tests, node i updates and forwards the packet to k and updates its PCT by adding a new entry.

4.4. Action by the Destination Node

When the destination node i receives the CDP, m, node i checks if *conn_id(m)* appears in one of its CRTs. If yes, node i updates the CRT by filling a new entry based on the information provided in this CDP. Otherwise, node i creates a new CRT for this connection request, and sets a timer which is no less than the average link delay multiplied by the hop-count limit. Upon expiration of the timer, any outstanding CDP corresponding to this connection request is no longer valid and has been discarded by some intermediate node, then node i starts the route selection process and then the route confirmation process.

Among all the candidate routes listed in a CRT, only those with $primary\_flag = 1$ might be selected as the primary channel route. The destination node chooses the shortest route (i.e., the one with the smallest hop_count value) to be the primary channel route. All the remaining candidate routes in the CRT are eligible to be the backup channel route, and the destination node chooses the shortest one that minimally overlaps with the primary channel route. The destination node starts the route confirmation process for both primary and backup channels simultaneously.

5. Backup Multiplexing

A node's attempt to choose a backup route without any conflict may not always be successful. It is therefore possible to activate more than one backup when a link failure occurs. For example, let $APLV_1 = (0,1,2,1,2)$. Then, if L_3 or L_5 fails, two DR-connections will attempt to activate their backups through L_1. If the spare resources reserved on L_1 can accommodate only one of the two DR-connections, one of the two will fail to activate its backup. To handle the case, the DR-connection manager responsible for L_i has to reserve more spare resources. The conflicting backups are not multiplexed over the same spare resources.

The DR-connection manager for a link checks if more spare resources need to be reserved using the APLV and SC of the link. Since all DR-connections are assumed to require an identical amount of bandwidth, SC_i can be calculated by dividing the total spare bandwidth reserved on L_i by the bandwidth of a DR-connection. If any element of $APLV_i$ is larger than SC_i, at least two conflicting backups are multiplexed on the same spare resources. In this case, it is necessary to reserve more spare resources.

When a node receives a backup-setup request, the DR-connection manager of the node updates the $APLV$ of the link that the backup traverses using the

$LSET$ included in the request of the corresponding primary. Using the new $APLV$, the DR-connection manager can determine if it will multiplex the new backup on the current spare resources or if it will reserve more resources.

A DR-connection manager may not be able to increase spare resources due to the shortage of resources, even when the new backup has conflicts with other backups. In such a case, we have two choices: (1) reject the request, or (2) multiplex the new backup on the previously-reserved spare resources with other backups that the new backup has conflicts with. We opt to take the second approach. Although multiplexing conflicting backups degrades fault-tolerance, there is a chance that one of the conflicting backups may be rejected, or one of the primary channels on the same link may terminate before a link failure that will trigger activation of conflicting backups. If a primary channel is released, its resources will be returned to the pool of free resources, and the DR-connection managers assign these free resources to spare resources.

6. Simulation and Analysis

We have conducted an in-depth simulation study to comparatively evaluate the three proposed routing schemes in terms of fault-tolerance and overhead. In this study, we measured the probability of successfully establishing a DR-connection, and the fault-tolerance of established connections under various load conditions and network configurations. We also evaluated the routing overhead for discovering backup routes and their resource overhead.

6.1. The Simulation Model

To investigate the performance of the proposed routing schemes under different network configurations, we selected networks with 60 nodes and the average node-degrees (E) of 3 and 4. The networks are generated by using the **Waxman** topology generator [11]. Each node acts as a router or switch, and links are assumed to be bi-directional, with an identical bandwidth capacity (C) in both directions.

parameters	value
N (number of nodes)	60
E (average node degree)	3,4
C (link bandwidth capacity)	100 Mbps
λ (connection request rate)	$\{0.2, 0.3 \cdots, 1.0\}$/sec
bw_req (bandwidth request)	2.5 Mbps
t_req (connection lifetime)	$20 \le$ t_req < 60 min

Table 1. The simulation parameters

The simulation study uses two traffic patterns. One, UT, is uniform random selection of source and destination nodes. The other, NT, is random pre-selection of 10 nodes as destinations for 50% of DR-connections. For simplicity, we assume that DR-connection requests arrive as a Poisson process with rate λ. Instead of using more realistic traffic models, we only consider simple traffic patterns, because our goal is to comparatively evaluate the proposed routing schemes, as opposed to providing absolute performance figures. Moreover, we assume that each connection requires a constant bandwidth (*bw_req*) and has a uniformly-distributed lifetime, *t_req*, between 20 and 60 minutes. The *network load* is defined as the total bandwidth reserved for all active real-time connections. Since we fix *bw_req* and *t_req* as constants, the network load depends only on the network capacity and the request arrival rate λ. The relevant parameters of the simulation are listed in Table 1, and the values are selected while keeping in mind the bandwidth and time constraints of typical video and audio applications.

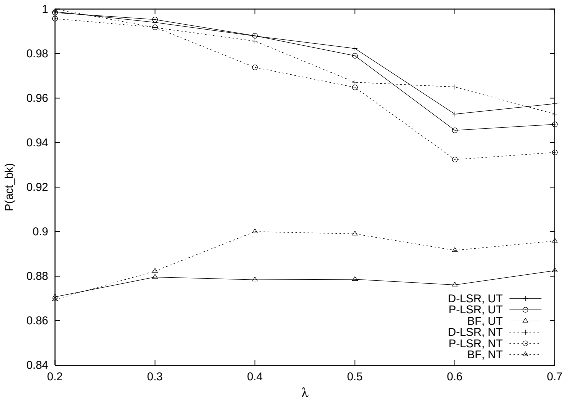
In the simulation study, we use scenario files to record the connection request and release events under various *bw_req* and λ values, and compare the performance of the proposed schemes by simulating them using the same scenario file. The scenario files are generated by **Matlab**, and the proposed routing schemes are implemented in, and simulated with, **ns**.

6.2. Performance Comparison

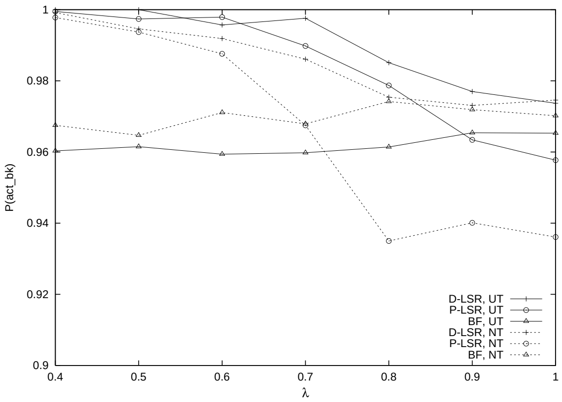
We selected four parameters, $\rho = \alpha = 1$, $\mu = 2$, and $\beta = 0$ for the bounded flooding scheme by simulation since increasing the flooding area larger than this barely improves the performance. We compared three routing schemes for different request arrival rates $\lambda \in \{0.2, 0.3, \cdots, 1.0\}$, under various network configurations. For convenience, in the following discussions, the symbol BF is used for the bounded flooding scheme. The fault-tolerance capabilities of the three routing schemes are plotted in Figure 4 and the *capacity overhead* in Figure 5, while varying traffic patterns and arrival rates.

D-LSR offers the best fault-tolerance among all the cases considered and BF the least in most cases. This was expected since D-LSR employs the largest amount of information about network status, and BF uses the most limited information.

The fault-tolerance of both D-LSR and P-LSR degrades as the network load increases for the following reason. Some of the backups chosen by D-LSR or P-LSR traverse longer paths to go around those links that have backup conflicts. Longer backups may generate conflicts with other backups established later.

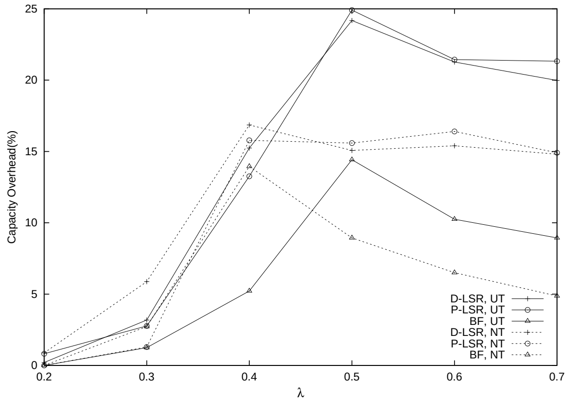


(a) $E = 3$

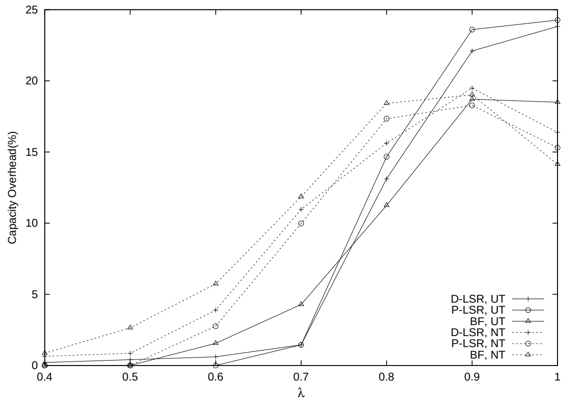


(b) $E = 4$

Figure 4. Fault-tolerance



(a) $E = 3$



(b) $E = 4$

Figure 5. Capacity overhead

This negative effect of longer backups is apparent when the network load is high, since the more requests for other backups will arrive before the longer backup is rejected when the arrival rate is high. BF does not show this phenomenon because backup route lengths are restricted by the bounding scope.

All three routing schemes provided higher fault-tolerance, as shown in Figure 4, when the network connectivity, E, is high. When the network has more links, there are more paths between any two nodes. Thus, in a highly-connected network, a node has more candidates for a backup and is more likely to find a backup that has less conflicts. Moreover, when the network connectivity is high, path selection is less critical to fault-tolerance. Since there are many candidate routes, even random selection can find a backup route with small conflicts.

As shown in Figure 4, when some DR-connections are concentrated on a small number of nodes, the performance gap between D-LSR and P-LSR is more pronounced. In such a case, some links may have many backups while others have very few. If a node should select one of two congested links, P-LSR cannot distinguish one from the other, since $\|APLV\|_1$ does not provide sufficiently-detailed information.

A network is said to be *saturated* if all of its resources are allocated to DR-connections. A saturated network cannot accept any more connections until some of the active connections terminate and release their resources. The simulated network gets saturated as λ reaches 0.5 (0.9) for the case of $E = 3$ ($E = 4$). To measure resource overheads of the three proposed routing schemes, we define the difference between the number of D-connections without backups and that of each routing scheme as *capacity overhead*. Since resources are reserved for backup channels of DR-connections, the number of DR-connections drops in the saturated network. Thus, the amount of resources reserved for backups could be indicated by the percentage of decreased number of connections that can be accommodated under each of the three routing schemes. As shown in the figure, all of the three proposed routing schemes decrease the network utilization by at most 25% when the traffic pattern is uniform, UT, and 20% when the traffic pattern is not uniform, NT. Recall that, without backup multiplexing, the network utilization would be decreased by 50% or more. What is more important is that DR-connections are shown to have high fault-tolerance and low capacity overhead until the network load reaches 70% of the maximum load.

We summarize the evaluation results as follows: *(1) multiplexed backup channels improve the fault-tolerance at the expense of slightly decreasing the network utilization, and (2) the lower the network connectivity, the more sophisticated routing algorithm is necessary.*

7. Conclusion

In this paper, we proposed and evaluated three routing schemes to find routes for the primary and backup channels of a dependable real-time connection. Two different methods are introduced to expand the link-state database in order to include the information about active real-time connections. Two link-state routing schemes discover backup routes with a high level of fault-tolerance at the expense of overhead to maintain additional information in the expanded link-state database. By contrast, the bounded flooding scheme does not require distribution and maintenance of link-state information, nor on-line route computation (e.g., the Dijkstra's algorithm). Instead, it is an on-demand scheme, and upon request of a real-time connection, the *qualified* routes are discovered by flooding the special channel-discovery packets within a bounded area.

Using extensive simulations with **Matlab** and **ns**, we evaluated the three routing schemes in terms of fault-tolerance and resource-capacity overhead. Our simulation results show that good fault-tolerance can be achieved at a reasonable decrease in the number of real-time connections that can be admitted. In addition, when the network load is not very high, allocation of spare resources for backup channels does not reduce the number of real-time connections that the network can accommodate.

References

[1] A. Banerjea. Simulation study of the capacity effects of dispersity routing for fault tolerance real-time channels. In *Proceedings of ACM SIGCOMM'96*, pages 192–205, Stanford, CA, 1996.

[2] A. Banerjea. Fault recovery for guaranteed perfoemance communications connections. *IEEE Transactions on Networking*, 7(5):653–668, Oct. 1999.

[3] A. Banerjea, C. Parris, and D. Ferrari. Recovering guaranteed performance service connections from single and multiple faults. In *Proceedings of IEEE GLOBECOM'94*, pages 162–168, San Francisco, CA, 1994.

[4] C. Dovrolis and P. Ramanathan. Resource aggregation for fault tolerance in integrated services networks. *Computer Communication Review*, 28(2):39–53, Apr. 1998.

[5] S. Han and K. G. Shin. Efficient spare resource allocation for fast restoration of real-time channels from network component failures. In *Proceedings of IEEE RTSS'97*, pages 99–108, 1997.

[6] S. Han and K. G. Shin. Fast restoration of real-time communication service from component failures in multihop networks. In *Proceedings of ACM SIGCOMM'97*, pages 77–88, 1997.

[7] S. Han and K. G. Shin. A primary-backup channel approach to dependable real-time communication in multi-hop networks. *IEEE Transactions on Computers*, 47(1), 1998.

[8] B. Kao, H. Garcia-Molina, and D. Barbara. Aggressive transmissions of short messages over redundant paths. *IEEE Transactions on Parallel and Distributed Systems*, 5(10):1044–1056, Jan. 1994.

[9] S. K. Kweon and K. G. Shin. Distributed QoS routing using bounded flooding. Technical Report CSE-TR-388-99, University of Michigan, 1999.

[10] P. Ramanathan and K. G. Shin. Delivery of time-critical messages using a multiple copy approach. *IEEE Transactions on Computer Systems*, 10(2):144–166, May 1992.

[11] B. M. Waxman. Routing of multipoint connection. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, Dec. 1988.

[12] Q. Zheng and K. G. Shin. Fault-tolerant real-time communication in distributed computing systems. In *Proceedings of IEEE FTCS'92*, pages 86–93, 1992.

[13] Q. Zheng and K. G. Shin. Establishment of isolated failure immune real-time channels in harts. *IEEE Transactions on Parallel and Distributed Systems*, 6(2):113–119, Feb. 1995.