Fast Restoration of Real-Time Communication Service from Component Failures in Multi-hop Networks

Seungjae Han and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109–2122, U.S.A.
{sjhan, kgshin}@eecs.umich.edu

Abstract

For many applications it is important to provide communication services with guaranteed timeliness and fault-tolerance at an acceptable level of overhead. In this paper, we present a scheme for restoring real-time channels, each with guaranteed timeliness, from component failures in multi-hop networks. To ensure fast/guaranteed recovery, *backup channels* are set up *a priori* in addition to each *primary channel*. That is, a *dependable real-time connection* consists of a primary channel and one or more backup channels. If a primary channel fails, one of its backup channels is activated to become a new primary channel. We describe a protocol which provides an integrated solution to the failure-recovery problem (i.e., channel switching, resource re-allocation, ...). We also present a resource sharing method that significantly reduces the overhead of backup channels. The simulation results show that good coverage (in recovering from failures) can be achieved with about 30% degradation in network utilization under a reasonable failure condition. Moreover, the fault-tolerance level of *each* dependable connection can be controlled, independently of other connections, to reflect its criticality.

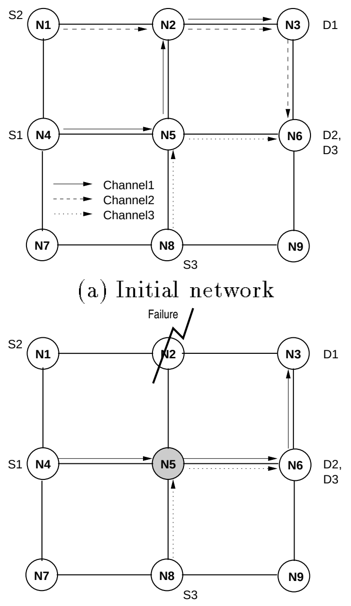
1 Introduction

Real-time communication services have become essential for many applications like digital continuous media (audio and motion video) and distributed real-time control. Unlike traditional datagram services in which average performance is of prime interest, guaranteeing such "quality of service" (QoS) as message delay and error rate is the key requirement of real-time communication services. In recent years, considerable efforts have been made to provide the timeliness QoS guarantee, while the importance of guaranteeing fault-tolerance QoS has been far less recognized. The survey paper by Aras *et al.* [ARA94] discusses many of existing real-time communication schemes. However, there are growing needs for communication services with a guaranteed level of fault-tolerance in many real-time applications. Suppose, for example, there is a very important video conference and network failures disconnect one or more participants from the conference for an unpredictably long period. This may lead to a failure or delay in reaching important strategic decisions, which can cause a significant economic loss. Catastrophic social consequences have actually been witnessed in recent breakdowns of the US telecommunication network.

Interestingly, most real-time communication schemes for multi-hop networks share three common properties: *QoS-contracted*, *connection-oriented*, and *reservation-based*. Essentially, a contract between a client and the network is established before actual message transfer. To this end, the client must first specify his input traffic behavior and required QoS. Then, the network computes the resource needs (e.g., link & CPU bandwidths, and buffer space) from this information, selects a path, and reserves necessary resources along the path. (If there are not enough resources to meet the QoS requirement, the client's request is rejected.) The client's messages are transported only via the selected path with resources reserved, and this virtual circuit is often called a *real-time channel*.

While this reservation-based approach has been successful in providing 'hard' guarantees on timeliness QoS, it causes a serious difficulty in achieving fault-tolerance. Traditional failure-handling techniques for datagram services are inadequate, because a real-time message can traverse only the path on which resources are reserved *a priori* for it and hence cannot be detoured around failed components on the fly. Instead, a new channel which does not use the failed components should be established before resuming the data transfer. However, establishing a new channel is usually a time-consuming process, which can result in a long service disruption. Moreover, such an approach cannot make any guarantee on successful failure recovery, because there may not exist a proper detouring path. Figure 1 illustrates such a situation.

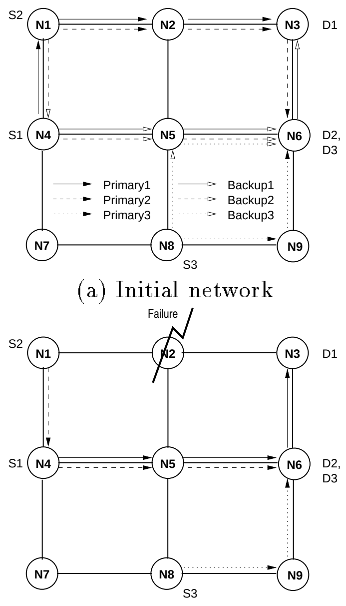
Figure 1 (a) shows a network which contains three real-time channels. Assume that two network nodes are connected by two simplex links, each of which can accommodate up to two channels. When node N2 fails, channels 1 and 2 need to be detoured around N2. Both channels may need to use shortest possible paths in order to maximize the chance of meeting their timeliness QoS requirements. As a result, the resource needs on the link from N5 to N6 exceed its capacity, and the link can accommodate only one of them, say channel 1, as shown in Figure 1 (b). Now, channel 2 has to be rerouted over a longer path. If channel 2's QoS requirement is too tight to fit the longer path, channel 2 cannot be recovered from N2's failure. An option is moving chan-



(a) Initial network

(b) After failure recovery

Figure 1: Failure recovery by blind rerouting



(a) Initial network

(b) After failure recovery

Figure 2: Failure recovery by the proposed scheme

nel 3 to a different path in order to accommodate channel 2 at the link from N5 to N6. However, this is not a good idea, since moving the existing channels can cause domino effects without guaranteeing successful rerouting of the affected channels. A better solution is not to set up channel 3 over the link from N5 to N6 in the initial network.

In this paper, we propose an efficient scheme to quickly restore real-time channels from network component failures. To assure successful rerouting and avoid the time-consuming channel re-establishment process, *backup channels* are set up *a priori* in addition to each *primary channel*. That is, a *dependable real-time connection* (or a $\mathcal{D}$-connection for short) consists of a primary channel and one or more backup channels.

A backup channel remains a cold-standby until it is activated. In other words, it does not carry any data in a normal situation, so that the resources reserved for the backup channel may be used by other traffic.[1] However, backup channels degrade the network's capability of accommodating real-time channels. If the application requires high-volume message streams (e.g., motion video), the degradation will become serious. To cope with this problem, we have developed a resource-sharing method, called *backup multiplexing*, in which resources are shared among backup channels in such a way that fault-tolerance is not compromised.

Figure 2 illustrates how the same failure in Figure 1 is handled in our scheme. Note the difference between the initial channel setups. In Figure 2 (a), primary-3 is routed over N9 instead of N5, because of the resource shortage on the link from N5 to N6. On that link, backup-3 is multiplexed with backup-1 and backup-2. In this example, we assumed that channels are established in the ascending order of their indices, using a shortest-path routing method.

The rest of the paper is organized as follows. Section 2 presents our design goals. Sections 3 and 4 describe, respectively, the connection-establishment and failure-handling procedures of the proposed scheme. Section 5 analyzes the service-disruption time caused by failures. Section 6 addresses the scalability issue. Section 7 presents the simulation results, demonstrating the efficiency of the proposed scheme. Section 8 discusses related work giving a comparative perspective, and the paper concludes with Section 9.

[1]Not only non-real-time traffic but also other real-time traffic can utilize the resources reserved for backup channels, if the underlying real-time channel scheme has the capability of dynamic QoS control like the layered transmission method [MCC96].

2 Design Goals

A real-time channel is a uni-directional one-to-one virtual circuit with the capability of timeliness-guaranteed, in-order, but unreliable message delivery. A real-time channel service is usually implemented with two protocols: Real-time Network Manager Protocol (RNMP) and Real-time Message Transmission Protocol (RMTP). The main function of RNMP is channel establishment and teardown, while that of RMTP is runtime control such as traffic shaping and message scheduling.

When a client requests a real-time channel to be established, he has to specify his traffic-parameters (e.g., maximum message rate) and QoS requirements (e.g., message delay bound). Using this information, RNMP performs an 'admission test,' which checks the availability of the resources necessary to meet the channel's QoS requirement. RNMP reserves resources if the admission test is positive. In RMTP, a traffic regulator is used to smooth (oftentimes bursty) packet arrivals, and one or multiple output queues are serviced for message scheduling and transmission. RMTP is closely related to RNMP, because the admission control of RNMP assumes a certain message-scheduling policy for RMTP.

The main intent of this paper is to develop a protocol which augments the existing real-time channel service with the fault-tolerance capability in multi-hop networks. To provide a fault-tolerant service, we must first define the underlying failure model. We assume that (infrequent) transient packet losses are acceptable to the target applications, or are dealt with by other techniques like forward error correction. Our scheme restores the real-time channel service which is disabled by "persistent" or "permanent" failures, e.g., *crash failures*.[2]

The proposed protocol, named Backup Channel Protocol (BCP), establishes $\mathcal{D}$-connections, reports detected failures to the nodes which are responsible for recovery operations, activates backup channels, resumes the disrupted real-time channel service, and reconfigures resources to cope with future failures (BCP does not deal with failure detection). There are five goals that drive the design of BCP:

- **Per-connection fault-tolerance control:** Each $\mathcal{D}$-connection is allowed to have a different fault-tolerance capability depending on its criticality. Unless the number and type of failures occurred exceed the fault-tolerance capability of the connection, a successful recovery is guaranteed.
- **Fast (time-bounded) failure recovery:** The service-disruption time of a $\mathcal{D}$-connection caused by failures is very short, and is bounded if certain conditions are met.
- **Robust failure handling:** Failures are always handled properly regardless of the number of their occurrences, and the QoS of nonfaulty real-time channels is not affected at all.
- **Small fault-tolerance overhead:** The amount of the additional resources required for fast/guaranteed recovery is acceptably small.
- **Interoperability/scalability:** The BCP can be placed on top of any real-time channel protocol, so it can be used in wide-area networks equipped with various (heterogeneous) protocols. Also, the BCP scales well, since it doesn't require each node to maintain global knowledge of network status.

3 Establishment of a $\mathcal{D}$-Connection

Instead of providing the same uniform level of fault-tolerance to all connections, we allow each client to specify his fault-tolerance QoS requirement. BCP then establishes necessary backups to meet the QoS requirement. Described below are the client interface and the channel-establishment procedure of BCP.

3.1 Fault-Tolerance QoS Parameter, $\mathcal{P}_r$

$\mathcal{P}_r$ represents the reliability of a $\mathcal{D}$-connection. Generally, the reliability of a system, denoted by $R(t)$, is defined as the probability that the system provides the required service from time 0 to t. In our case, the required service (i.e., fault-tolerant real-time channel service) will be provided unless all channels of a $\mathcal{D}$-connection fail (near) simultaneously.

Let's consider how to derive $R(t)$ of a $\mathcal{D}$-connection. Assuming a Poisson failure process with rate λ, we derive $R(t)$ of each network component to be $e^{-\lambda t}$. For the convenience of presentation, we further assume that the failure rates of all network components are same and all failures are statistically independent. Then, $R(t)$ of a channel can be expressed as $e^{-n\lambda t}$, where the channel path consists of n components.

[2]If a system halts and remains halted, it is said to have crashed. It can or cannot be recovered by restarting (rebooting) the system. A link can crash by losing all messages transmitted over it.

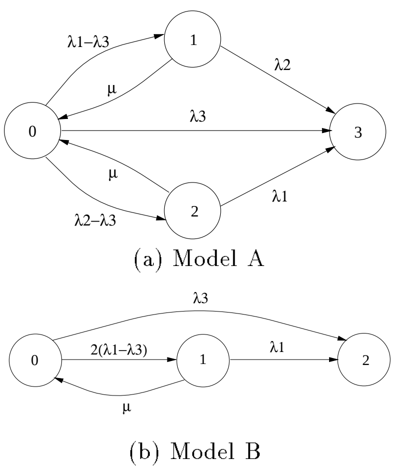


Figure 3: Markov models to derive $R(t)$

In other words, the failure rate of the channel is $n\lambda$. Finally, $R(t)$ of a $\mathcal{D}$-connection can be modeled with a Markov process using the failure rates of its channels. For example, Figure 3 (a) shows a continuous-time Markov model to derive $R(t)$ of a $\mathcal{D}$-connection with a single backup channel, where μ is the channel repair (or re-establishment) rate, λ_1 and λ_2 are failure rates of the primary and backup channels, respectively, and λ_3 is the failure rate of the shared part of both channels. State 0 is the initial state and state 3 is the absorbing state. Figure 3 (b) is a simplified model when the primary and backup channels are of the same length. Using the technique in [TRI82], one can calculate $R(t)$ of a $\mathcal{D}$-connection from these Markov models.[3]

However, representing the QoS parameter as a function of time is unsuitable for the client interface model. Furthermore, the channel repair rate (μ) is much larger than the channel failure rate — the channel re-establishment time is in the order of seconds or minutes, whereas MTBF is in the order of 1000 hours. Thus, the system returns to the initial state quickly unless the second failure occurs near-simultaneously. Based on these observations, instead of using Markov models, we use a combinatorial model in which each network component is assigned a probability, λ, of failure during one time unit and the system (i.e., a $\mathcal{D}$-connection) is reset to the initial state at the start of each time unit. So, $\mathcal{P}_r$ is equal to the probability that at least one channel of the $\mathcal{D}$-connection remains healthy during one time unit. For example, the $\mathcal{P}_r$ of a $\mathcal{D}$-connection with a single backup is $P(\text{primary not fail}) + P(\text{primary fails} \cap \text{ backup not fail})$.

3.2 Backup Multiplexing

The $\mathcal{P}_r$ of a $\mathcal{D}$-connection is determined mainly by the number and the routing of its backup channels. A connection with more backups will have a higher probability of at least one of its backups surviving network failures. The links/nodes used by a primary channel may preferably be avoided in routing its backups, because overlapping routes among the channels of the same $\mathcal{D}$-connection will degrade its fault-tolerance QoS.

As far as actual resource consumption is concerned, a backup channel costs nothing, since it does not actually transport any information until it is activated. However,

[3]$R(t) = 1 - P(\text{the system is in the absorbing state at time } t)$.

a backup channel is not free, as it requires the same amount of resources as its primary channel to be reserved, for immediate activation upon failure of the primary. As a result, equipping each $\mathcal{D}$-connection with a single backup routed disjointly with its primary reduces the network capacity by 50% or more. We call the resources reserved for backups '*spare resources.*' The large amount of spare resources can seriously degrade the attractiveness of the backup-channel scheme.

To alleviate this problem, we have developed a resource sharing technique, called *backup multiplexing*. Its basic idea is that, at each link, we reserve only a very small fraction of spare resources[4] needed for all backups going through the link. That is, resources for backup channels are 'overbooked.' One of the key problems in backup multiplexing is to decide which backups will share the same resources. A natural solution to this problem is to choose those backups which are less likely to be activated simultaneously. The probability of simultaneous activation of two backups belonging to two different $\mathcal{D}$-connections is bounded by the probability of simultaneous failure of their respective primary channels. This probability depends on the routing of the primary channels, and increases with the number of components shared between the primary channels.

For each link, we calculate the probability — denoted by $\mathcal{S}(B_i, B_j)$ — of simultaneous activation of two backups, B_i and B_j, whose primary channels are M_i and M_j, respectively. Assuming that failures occur independently with the same probability λ, we get:

$$\begin{aligned}
\mathcal{S}(B_i, B_j) &= 1 - P(\text{no failure in shared components}) \\
&\quad \cdot P(\text{no simultaneous failures in the rest}) \\
&= 1 - (1-\lambda)^{sc(M_i,M_j)} \cdot \{1 - (1 - (1-\lambda)^{c(M_i)-sc(M_i,M_j)}) \\
&\quad \cdot (1 - (1-\lambda)^{c(M_j)-sc(M_i,M_j)})\} \\
&= 1 - \{(1-\lambda)^{c(M_i)} + (1-\lambda)^{c(M_j)} \\
&\quad - (1-\lambda)^{c(M_j)+c(M_i)-sc(M_i,M_j)}\}
\end{aligned}$$

where $c(M_i)$ and $c(M_j)$ are the component counts in M_i and M_j, respectively, and $sc(M_i, M_j)$ is the number of components shared between them. Here, components include both nodes and links. One can use different failure rates for nodes and links by slightly modifying the equation.

Based on this probability, the set of backups to be multiplexed together is determined for each backup on each link, i.e., multiplexing is done hop-by-hop. B_i and B_j are multiplexed if $\mathcal{S}(B_i, B_j)$ is smaller than a certain threshold ν, called the *multiplexing degree*, which is specific to each backup. The smaller ν of a backup, the higher fault-tolerance will result. For instance, if ν for a backup B_i is set to λ, fast recovery of the corresponding $\mathcal{D}$-connection from any single node/link failure is guaranteed, because B_i will not be multiplexed with any other backup whose primary overlaps with M_i. This way, per-connection control of fault-tolerance is possible, thus allowing more important connections to have higher fault-tolerance (e.g., tolerating harsher failures). In this paper, each backup is required to have the same multiplexing degree on all of its links for ease in managing $\mathcal{P}_r$.

Let $\Pi_{B_i,\ell} = \{B_\alpha, B_\beta, \ldots\}$ denote the set of backups which are not multiplexed with B_i on link ℓ. One way to determine the spare resources at link ℓ is to find the highest resource requirement among all sets of $\{\Pi_{B_i,\ell} + B_i\}$, where all backups are considered equally regardless of their multiplexing degrees. This method may overestimate the amount of required spare resources at a link, when there are multiple backups with different multiplexing degrees running over the link. Suppose there are one backup with a very small ν and many backups with large ν on a link. Then, Π_ℓ of the backup with a very small ν will determine the amount of spare resources at the link, which may be much larger than actually needed. To get around this problem, we consider only backups with no greater multiplexing degrees than that of B_i when $\Pi_{B_i,\ell}$ constructed.

[4]In this paper, we consider only link bandwidth for simplicity, but other resources like buffer and CPU can be treated similarly.

3.3 Calculation of $\mathcal{P}_r$ with Backup Multiplexing

When a backup channel is activated, it draws necessary resources from the spare resources. Since multiplexing is based on probabilistic relations, there is a possibility, albeit rare, that the backups which are multiplexed together need to be activated simultaneously. Such unlikely activations can cause the exhaustion of spare resources, so that the remaining backups cannot be activated; "*multiplexing failures*" are said to occur to those backups.

When we calculate the $\mathcal{P}_r$ of a $\mathcal{D}$-connection with backup multiplexing, we have to consider the possibility of multiplexing failures. The $\mathcal{P}_r$ of a $\mathcal{D}$-connection with a single backup B_i routed disjointly with the primary channel M_i is

$$\begin{aligned}
\mathcal{P}_r &= P(M_i \text{ not fail}) + \\
&\quad P(M_i \text{ fails}) \cdot P(B_i \text{ not fail}) \cdot \{1 - P_{muxf}(B_i)\},
\end{aligned}$$

where $P_{muxf}(B_i)$ represents the probability that B_i is not available due to a multiplexing failure. $\mathcal{P}_r$ with more backups can be derived in a similar way.

For simplicity, we derive an upper-bound of $P_{muxf}(B_i)$. If any backup is multiplexed with B_i, the ν value of B_i is always greater than the probability of simultaneously activating B_i and the backup. In other words, the probability of avoiding a multiplexing failure is at least $1 - \nu$. Therefore, the probability that B_i will suffer from a multiplexing failure on link ℓ is not greater than $1 - (1-\nu)^{|\Psi_{B_i,\ell}|}$, where $|\Psi_{B_i,\ell}|$ is the number of backups multiplexed with B_i on link ℓ (i.e., $\Psi_{B_i,\ell} = \{\textit{all backups on } \ell\} - \Pi_{B_i,\ell} - B_i$). Considering all the links B_i runs through, we get

$$P_{muxf}(B_i) \leq \sum_{all\ \ell\ of\ B_i} 1 - (1-\nu)^{|\Psi_{B_i,\ell}|}.$$

3.4 Backup Channel Establishment

As in the case of primary channels, route selection and resource reservation are crucial steps of backup channel establishment. Here we use the shortest-path routing to select backup paths. Algorithms to find multiple disjoint shortest paths between two nodes are given in [WHA90, SID91].

As to spare resource reservation, there can be various QoS-negotiation schemes between clients and BCP to determine the number of backups and the associated multiplexing degrees. We describe two possible schemes. In the first scheme, the client-specified $\mathcal{P}_r$ requirement is met "loosely," as opposed to "literally." At first, BCP selects the number of backup channels and their multiplexing degrees by considering the $\mathcal{P}_r$ requirement and/or the network status. Then, the backups are established and the resultant $\mathcal{P}_r$ of the connection calculated. If backup establishment is successfully completed, the resultant $\mathcal{P}_r$ is notified to the client. The client may or may not be satisfied with the offered fault-tolerance level, and may accept or reject the offer. If the backups cannot be established due to insufficient resources available, the client's request will be rejected. (The rejected client may opt to retry with a lower $\mathcal{P}_r$ requirement.)

In the second scheme, the client's $\mathcal{P}_r$ requirement is met as requested. In contrast to the first scheme, BCP establishes backup channels incrementally until the required $\mathcal{P}_r$ is achieved. Assume that the channel establishment is initiated by the source node.[5] A backup channel is established by using a pair of channel-establishment messages: (i) the 'resource reservation message' from source to destination and (ii) the 'resource relaxation message' from destination to source. In the forward pass (reservation message) to the destination, BCP reserves spare resources for the backup without multiplexing, while calculating the $|\Psi_{B_i,\ell}|$ of each link ℓ on the channel route with various ν values. The reservation message collects the $|\Psi_{B_i,\ell}|$ calculation results and passes them to the destination node. Then, the destination node selects the largest ν which satisfies the required $\mathcal{P}_r$ based on the collected information. Essentially, the problem of meeting the $\mathcal{P}_r$ requirement is transformed to that of deciding the multiplexing degree. Fortunately, we need to try only a couple of different ν values, because the values of $\mathcal{S}(B_i, B_j)$ are distributed around integer multiples of λ when λ is small, i.e., $\mathcal{S}(B_i, B_j) \approx c(M_i)\cdot\lambda + c(M_j)\cdot\lambda - \{c(M_i) + c(M_j) - sc(M_i, M_j)\}\cdot\lambda = sc(M_i, M_j)\cdot\lambda$. Thus, the backup channels on a link can be classified into a certain number of classes according to their multiplexing degrees which are discrete numbers. The number of classes are not greater than the length of the longest possible path in the network. In the backward pass (relaxation message) from destination to source, the spare resources on the channel path are multiplexed according to the selected ν. If the required $\mathcal{P}_r$ cannot be met with a single backup, additional backups are established. The multiplexing degree of the backups set up previously can be adjusted (further relaxed), if necessary. If the required $\mathcal{P}_r$ is too high to satisfy, $\mathcal{P}_r$ should be renegotiated.

For both schemes, the BCP daemon at each node has to maintain the information about each backup running through the node, including the path of its primary, the multiplexing threshold, the non-multiplexable channel set, and other information like the current channel state (which will be discussed in Section 4). We will discuss the complexity and scalability of the backup-channel establishment procedure in Section 6.

4 Failure Recovery Procedure

The first step in handling a failure is its detection. Depending on the semantic of a failure, all channels on the failed component may fail or only part of them may fail. QoS requirements can also make an impact on the manifestation of a failure. For instance, real-time channels for certain applications may not be able to tolerate an error rate which is acceptable to other channels set up for different applications. We have developed techniques for detecting channel failures, evaluated their efficiency experimentally, and reported the results in [HAN97a].

In this paper, we assume the existence of a proper failure-detection mechanism in which failed components are detected by their neighbor nodes, and focus on the procedure after failure detection.

[5] This is not a restriction. The destination can initiate the channel establishment.

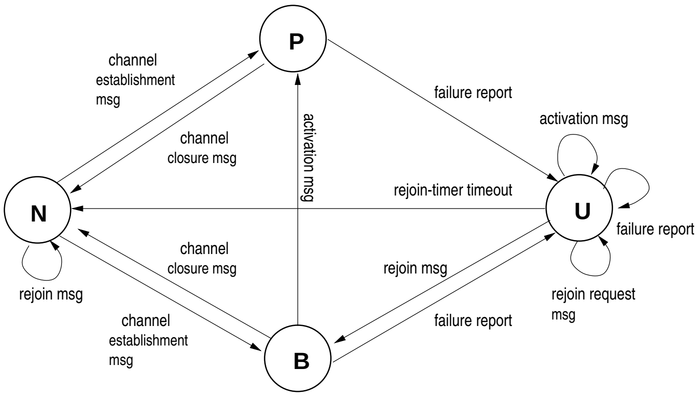


Figure 4: Channel state transition

4.1 Overview

If the node which detects a failure is different from the node which is responsible for channel switching, the failure should be reported to the latter node. There are three important issues in failure reporting. First, who will need to receive failure reports ? Second, which path will be used for failure reporting? Third, what information needs to be carried in failure reports? Our approach to these issues is as follows: (i) failure reports are sent from the failure-detecting nodes only to the end-nodes of failed channels, (ii) failure reports are delivered through healthy segments of the failed channels' paths, (iii) each failure report contains the 'channel-id' of the failure channel.

Our approach handles multiple (near-) simultaneous failures very naturally and easily. A failure report will be discarded by a node when the same report had already been received/passed through. Thus, if multiple failures occur to a channel, only one failure report will reach its end-nodes, and all the other reports will be lost due to the failures themselves or discarded by intermediate nodes.

When an end-node of a $\mathcal{D}$-connection receives a failure report on its primary channel, it selects one of its backups and sends an '*activation message*' along the path of the selected backup. To identify the health of backups, failures of backup channels are reported to their end-nodes in the same way as primary channel failures. During its journey, the activation message can come across a node which had already received a failure report of the backup being activated. In such a case, the activation message is simply discarded, because this new failure will be reported and another activation message will follow.

After activating a backup channel to become a new primary channel, BCP needs to reconfigure the resource reservation at the intermediate nodes of the new primary channel, because some resources shared with other backups are now dedicated to the new primary channel. If the spare resources at a link are exhausted by the activation, the remaining backup channels on the link cannot function as standby channels, i.e., multiplexing failures. Multiplexing failures are reported in the same way as component failures.

The key principle of our failure-recovery process is *localization*, so that the traffic on non-faulty parts of the network remains unaffected by failure recovery.

4.2 Failure Reporting & Backup Activation

The failure-recovery process outlined in the previous subsection is elaborated on with a state transition diagram in Figure 4. At each node, a channel can be in one of four states: non-existent state (N), healthy primary channel state (P), healthy backup channel state (B), and unhealthy channel state (U). The initial state is N. Upon reception of a '*channel-establishment message*,' the state machine enters state P or B. When a node receives a failure report (or detects a failure) in state P or B, the state machine enters U and the failure report is forwarded to the appropriate node. Additional failure reports received in state U are ignored. When an activation message is received in state B, the state machine enters P. The activation messages received in state U are ignored. The state transition for resource reconfiguration (e.g., from U to N, or from U to B) will be detailed later.

Now, we describe and compare schemes for failure reporting and backup activation. Figure 5 illustrates three schemes. The main distinction among these schemes is where the failure reports and activation messages are generated and destined for. In Scheme 1 (Figure 5 (a)), the downstream node of the failed component generates a failure report and sends it to the destination node of the failed channel. Then, the destination node initiates an activation message, which travels in the opposite direction of the backup channel to be activated. By contrast, in Scheme 2 (Figure 5 (b)), the upstream node generates the failure report, the channel source node receives the failure report, and the activation message is sent to the channel destination node. Scheme 3 (Figure 5 (c)) is a hybrid of the first two schemes. Both end-nodes of a failed channel receive failure reports, and backup-channel activation is done in both ways. If an activation message reaches a node on which the backup channel has already been activated by the activation message from the other end-node, the activation message is discarded by the node.

Scheme 2 and Scheme 3 have an advantage over Scheme 1 in terms of recovery delay, because data transfer through the new primary channel can be resumed immediately after sending the activation message,[6] while in Scheme 1 the data transfer has to wait until the activation message is received by the source node. If a failure occurs near the destination node, this advantage will be minimal.

Scheme 3 has an edge over Scheme 2 in two aspects. First, all nodes of a failed channel are informed of the failure, which is useful for resource reconfiguration. Second, the channel destination node can prepare early for channel switching, and the activation delay will be reduced by the bi-directional activation. If a $\mathcal{D}$-connection is equipped with multiple backups, it is necessary that both end-nodes activate the same backup.[7] One way to accomplish this is to allocate serial numbers to the backups of each $\mathcal{D}$-connection, and select a backup according to the serial number. In the remainder of this paper, we assume the use of Scheme 3.

4.3 Priority-based Activation

Connection priorities can be considered in the activation of backup channels. The idea is to activate the backups belonging to higher priority $\mathcal{D}$-connections ahead of those of lower priority $\mathcal{D}$-connections, if there are not enough resources to grant all activation requests.

[6] Albeit unlikely, if a data message arrives at intermediate nodes of the new primary channel before the channel is activated, the data message will be discarded with no harm.

[7] If the destination node activates a different backup, the backup need to be deactivated, since data messages have already been transmitted over the backup activated by the source node.

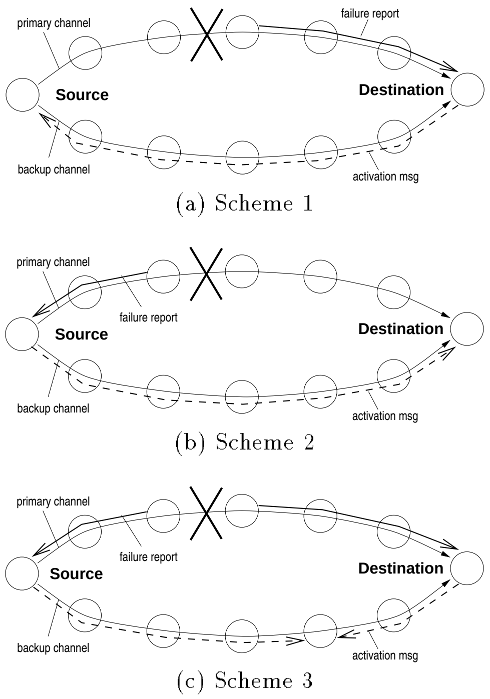


Figure 5: Channel-switching schemes

This priority-based activation can be achieved by delaying the activation of backups. In this method, an activation message is sent after a certain delay determined by the multiplexing degree of the backup channel to be activated. (Recall that the importance of a backup channel is represented by the multiplexing degree.) Thus, the activation of backups with a large multiplexing degree (i.e., lower-priority backups) is delayed so that the backups with small multiplexing degrees (i.e., higher-priority backups) may be activated first. The main shortcoming of this method is that the 'activation wait delay' is always imposed on lower-priority connections. To completely avoid priority inversion, this wait delay should be longer than the transmission delay of the activation message over the longest channel path in the network. In a large-scale network, the recovery delay incurred to lower-priority channels could be unacceptably long.

Another way is to allow a higher-priority backup to preempt lower-priority backups, if the lower-priority backups have already been activated and there are not enough spare resources for activating all of them. Preempted channels are handled as if they were disabled by component failures. So, the overhead associated with a preemption is the same as that for a failure recovery. Note that the recovery delays of lower-priority connections would be extended only if preemptions occur. An important issue of this method is the time granularity with which lower-priority connections can be preempted. If the preemptable interval is longer than the time needed for 'backup activation,' higher-priority backups will preempt active channels (i.e., primary channels of lower-priority connections). To avoid oscillation, the preemptable interval should be short, so that lower-priority connections may be preempted only by the higher-priority connections which fail (near-) simultaneously with them.

4.4 Resource Reconfiguration

After the disrupted service is resumed, the faulty channels will be torn down and, if necessary, new backup channels will be established. To tear down a channel, a '*channel-closure message*' is usually sent over the channel's path, so that resources for the channel may be released. However, if failures disconnect a channel's path or disable the channel end-nodes, the resource-release process becomes complicated. To facilitate the reclaiming of the resources of failed channels, we borrow the concept of "soft-state connections" in RSVP [ZHA93]. When an intermediate node of a channel receives a failure report (or detects a failure), it sets a *rejoin timer* whose expiration automatically triggers the channel teardown at the node.[8] Recall that in our failure reporting scheme (Scheme 3), all intermediate nodes either detect failures or receive failure reports, regardless of the number and location of failures. The purpose of the rejoin timer is to give the unhealthy channels (i.e., in U state) a chance to repair themselves. Channel repair can eliminate the need of establishing new channels, in case the unhealthy channels become usable again soon.

When the channel's source receives a failure report, it sends its destination a '*rejoin-request message*' via the path of the failed channel, and each healthy intermediate node forwards this message. If the failed component becomes healthy again before the rejoin timer expires, it will also forward the rejoin-request message. Otherwise, the rejoin-request message will not propagate beyond the failed component. If a (backup) channel enters U state because of a multiplexing failure, more spare resources have to be allocated to restore the channel. If it is impossible to allocate additional spare resources because of resource shortage, the rejoin-request message will be dropped.

If the channel destination node receives the rejoin-request message, the channel can be considered healthy (repaired). The destination node then sends a '*rejoin message*' back to the source node over the same path, and the channel state is changed from U to B, meaning that a repaired channel becomes a backup channel. If the rejoin timer had already expired when the rejoin message arrives at a node (i.e., in N state), the channel should be torn down as the resources for the channel had already been released. To undo the rejoin operations which have already done for the channel, a channel-closure message is generated by that node and is sent toward the channel destination. Figure 6 illustrates this case. The initial value of the rejoin timer should be chosen carefully. While it should be small for a quick teardown of unhealthy channels, it should also be large enough to allow their repair, including (i) the failure reporting delay, (ii) the round-trip time of the rejoin-request message and the rejoin message, (iii) the time for additional resource allocation.

If all channels of a $\mathcal{D}$-connection fail simultaneously, a new primary channel has to be established from scratch. When there is no route which can meet the QoS requirement of the $\mathcal{D}$-connection, its client will be informed of the unrecoverable failure. Similarly, if any channel end-node fails or the network is partitioned, all attempts of channel re-establishment will be unsuccessful and the client will be informed of the unrecoverable failure. In any of these cases,

[8]This operation is executed by BCP independently of the underlying RNMP.

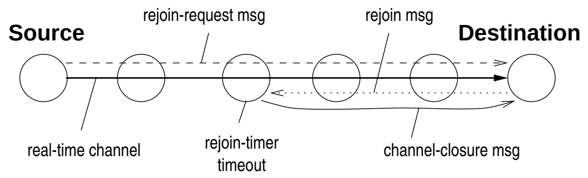


Figure 6: Repair/closure of an unhealthy channel

all the resources reserved for the connection will be released, when the rejoin timer expires.

So far, we have discussed the tear-down and repair of failed channels. Another issue which must be addressed is how to reconfigure spare resources after backups are activated. Because spare resources are shared among multiple backups, the activation of some backups can degrade the fault-tolerance capability of the remaining backups. The required spare resources should be recalculated, and additional resources should be reserved, if necessary, to preserve the fault-tolerance capability of the remaining backups. If the required spare resources are not available, some of the remaining backups have to be closed (and/or moved to different paths). Then, one has to determine which backups to close or move. The solution to this problem should account for the fault-tolerance QoS requirement of each connection, since a connection is vulnerable to failures during the re-establishment of its backups.

5 Bounded-Time Failure Recovery

Most of the resource reconfiguration operations, especially channel re-establishment, are time-consuming. However, unlike failure detection, failure reporting, or channel switching, resource reconfiguration is not a time-critical action, because its delay does not directly affect the service-disruption time except for the case of loss of all channels of a $\mathcal{D}$-connection.[9] If there is at least one backup surviving failures, we can avoid the channel re-establishment and achieve fast recovery.

The transmission delay of control messages, such as failure reports, is a major component of the recovery delay. The delay of such control messages is unpredictable, if they were transported as best-effort messages. Assigning the highest priority to control messages is not a good solution either, as it may affect the QoS of regular real-time communication services. Suppose there are malicious nodes or a large number of coincident failures. In such cases, the flood of control messages can paralyze the whole (or part of) network. To achieve fast and robust transmission of control messages, we use a special-purpose real-time channel, called the *real-time control channel* (RCC).

5.1 The RCC Network

An RCC is a single-hop real-time channel which connects two BCP daemons for the transmission of time-critical control messages. When the network is initialized BCP establishes a pair of RCCs, one in each direction, on every link of the network. RCCs will also be established, when failed components rejoin the network. The messages transferred over RCCs are called '*RCC messages.*'

[9]But resource-reconfiguration delay can influence the recovery capability/delay in handling future failures.

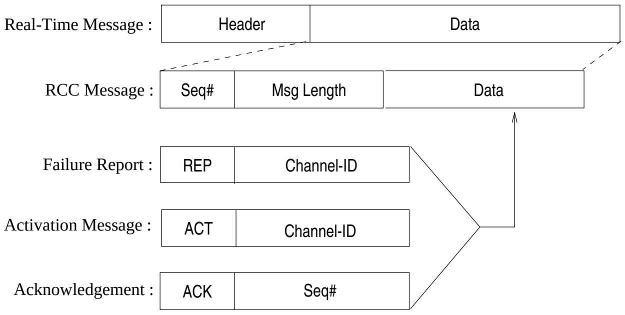


Figure 7: The RCC message format

The format of an RCC message is shown in Figure 7. Basically, an RCC message contains a combination of failure reports, activation messages, and acknowledgments. The control messages related to resource reconfiguration are excluded, since their delays are not time-critical. An interesting component of the RCC message format is acknowledgments, which are used to ensure reliable transmission of control messages. Generally, real-time communication does not support message retransmission, because there is not usually enough time for retransmission before the message deadline expires, and occasional losses of real-time (data) messages are tolerable in many applications. However, the loss of control messages is critical even in these applications. Each RCC message is acknowledged hop-by-hop between BCP daemons, and if a BCP daemon does not receive an acknowledgment of the RCC message which it sent, it resends the unacknowledged RCC message. Each RCC message contains the sequence number, so that duplicated messages may be easily detected and discarded.

While the exact specification depends on the underlying real-time channel protocol, we model an RCC by three parameters without loss of generality: maximum message size S_{max}^{RCC}, maximum message rate R_{max}^{RCC}, and maximum message delay D_{max}^{RCC}. RCC messages are transmitted as follows. Each RCC-message has its eligible time and is held until it becomes eligible for transmission. Thus, the minimum interval $(1/R_{max}^{RCC})$ is enforced between two RCC messages. Until the next time to transmit RCC messages, the BCP daemon at a node collects the outgoing control messages and forms RCC messages according to the destinations of the control messages. In the next node, the received RCC message is fragmented and new RCC messages are formed. The sequence of disassembly and assembly of RCC messages continues.

The collection of RCCs on all links forms a virtual network,[10] called the *RCC network*, of the same topology as the underlying physical network. One can consider a (physical) network as a composition of three logically separated networks — the primary-channel network, the backup-channel network, and the RCC network.

5.2 RCC Message Delay

The delay of control messages will depend directly on the capacity of the RCC network, i.e., if the capacity of the RCC on each link is large enough to accommodate all control messages on the link, the timely delivery of control messages can be guaranteed.

There is an upper bound on the control message traffic,

[10] A separate network in terms of resource reservation

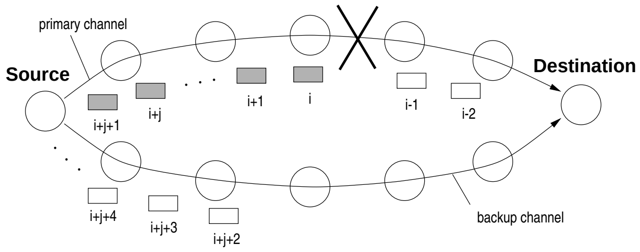


Figure 8: Message loss during failure recovery

for the reasons given below. The number of failure reports on a link ℓ cannot exceed the number of primary/backup channels on a pair of links between two nodes incident to ℓ. We have to consider both links, because failure reports for a channel can travel in both forward and backward directions of the channel, depending on the failure location. Similarly, the number of activation messages on link ℓ is bounded by the number of backup channels on the pair of links between the two nodes incident to ℓ. Since both the failure report and the activation message for the same channel cannot be transported over the same link at the same time, the control-message delay on any link is bounded by D_{max}^{RCC} if the following condition is met:

$$S_{max}^{RCC} \geq \max\{z : \ z = x \cdot y, \forall \text{ link pairs}\},$$

where $x =$ the size of a control message and $y =$ the number of channels on a link pair. If the maximum control message traffic on a certain link exceeds S_{max}^{RCC}, some control messages may experience a longer delay than D_{max}^{RCC} at that link.

5.3 Failure-Recovery Delay Bound

Now, let's consider the failure-recovery delay. We assume that the failures are immediately detected, control messages are delivered without loss/retransmission, and the computational delays for recovery operations are negligible compared to the control message delays. Then, the failure-recovery delay, Γ, is the sum of 'failure reporting delay' and 'activation retrial delay'. The delay for the activation message is not included in Γ, because services are resumed immediately after sending the activation message by the source node, assuming the activation message is delivered faster than the data message. If the control message delay on each link is bounded by D_{max}^{RCC}, we can derive an upper bound of Γ as follows.

The 'failure reporting delay' is less than $(\mathcal{K}-1)D_{max}^{RCC}$, where $\mathcal{K}$ is the number of hops of the longest-route channel of the $\mathcal{D}$-connection. The 'activation retrial delay' needs to be considered in case the connection has multiple backups. When the activation message for a backup encounters failures during its journey, one additional round-trip delay is added to the recovery delay: the transfer delay of the unsuccessful activation message itself and the delay for reporting the new failure. It is bounded by $2(b-1)(\mathcal{K}-1)D_{max}^{RCC}$ where b is the number of backups. With a single backup configuration, the failure-recovery delay of a $\mathcal{D}$-connection is equal to the failure reporting delay. If the failed component is located close to the source node, the recovery delay will be very short. Figure 8 illustrates the message loss during failure recovery (shaded messages are lost).

6 Scalability

The proposed scheme scales well because it does not require each node to maintain global knowledge of the network traffic conditions or to generate any type of messages to broadcast. Backup multiplexing is performed hop-by-hop, and therefore, at each link, only the knowledge of primary channels whose backups traverse the link is required. Such information can be easily collected, if a backup channel-establishment message carries the path information of its primary channel.[11] Control messages are sent only over the paths of channels affected by failures, instead of broadcasting them to the entire network.

The efficiency of backup multiplexing does not degrade as the network scales up. In fact, backup multiplexing will become more effective in large-scale and highly-connected networks, because such networks contain more versatile paths between two end nodes of a connection, thus lowering the probability that primary channels overlap with one another.

The delay for backup multiplexing does not directly affect the failure recovery delay, but the computational complexity of backup multiplexing is a matter of concern. Its essential part is the construction of a set of non-multiplexable backups, $\Pi_{B_i,\ell}$, on each link ℓ, taking $O(n)$ time, where n is the number of backup channels on link ℓ. (This is because each calculation of $\mathcal{S}(B_i, B_j)$ requires constant time.) To find the largest set, BCP needs to construct $\Pi_{B_i,\ell}$ for all backups on ℓ, which requires $O(n^2)$ time. However, if we store each $\Pi_{B_i,\ell}$ calculated before the new establishment request for B_j, we only need to update each $\Pi_{B_i,\ell}$ by calculating $\mathcal{S}(B_i, B_j)$. Hence, the complexity can be reduced to $O(n)$ at the expense of memory.

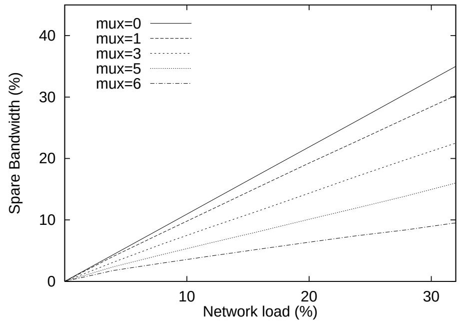
7 Evaluation

The proposed scheme is evaluated by simulating an 8×8 torus (wrapped mesh) network and an 8×8 mesh network. In these simulated networks, neighbor nodes are connected by two simplex links, one for each direction, and all links have an identical bandwidth. To obtain a similar total capacity for both networks, we set the link capacity of the torus network to 200 Mbps and set that of the mesh network to 300 Mbps.

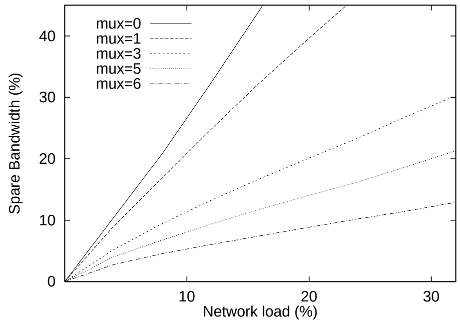
Channels of each $\mathcal{D}$-connection were routed disjointly by a sequential shortest-path search algorithm. Thus, the primary channel was routed first over a shortest path, then the backup was routed without using the components of the primary channel. For simplicity, the same traffic model was used for all channels, so each channel requires 1 Mbps of bandwidth on each link of its path. The end-to-end delay requirement of each channel is assumed to be met if the channel path is not longer than the shortest-possible path by more than 2 hops. A total of 4032 connections were established incrementally, so that there may exist a $\mathcal{D}$-connection between each node pair, i.e., $64 \cdot 63 = 4032$.

7.1 Spare Resource Overhead

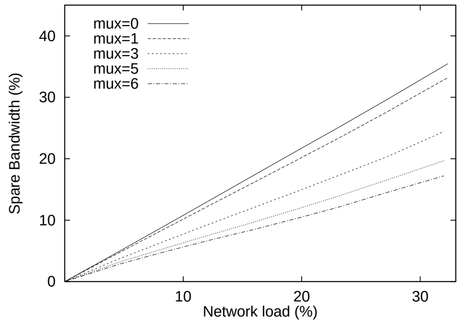
We first measured the average spare bandwidth for various backup configurations. For ease of comparison, all $\mathcal{D}$-connections are assumed to require the same number of backups and the same multiplexing degree. Single and double backup configurations were simulated in the torus network, but only the single backup configuration can be simulated in the mesh network because of its topological limitation. Seven different multiplexing degrees were applied in each case.



(a) Single backup in 8×8 torus



(b) Double backups in 8×8 torus



(c) Single backup in 8×8 mesh

Figure 9: Average spare-bandwidth reservation

Figure 9 shows the simulation results. The 'network-load' is a metric to indicate the ratio of the total bandwidth consumed by all primary channels to the total network bandwidth capacity. The establishment of 4032 connections resulted in a $33 \sim 34\%$ network-load in both networks.

The notation 'mux=α' means that two backups are multiplexed when their primary channels share less than α network components, i.e., $\nu = \alpha\lambda$. ('mux=0' implies that multiplexing was disabled). The results of 'mux=2' and 'mux=4' were not plotted in Figure 9, because, due to the nature of channel routing, they were very close to the cases of 'mux=3' and 'mux=5', respectively. Thus, two channel paths are not likely to share two nodes without sharing a link between the nodes, so the results of 'mux=2' and 'mux=3' are very close to each other. The case of sharing two consecutive links (i.e.,'mux=4' and 'mux=5') can be reasoned similarly.

There are several interesting observations to make from

[11] Assuming that a backup is established after its primary has been routed.

Muxing degree	mux=1	mux=3	mux=5	mux=6
Spare bandwidth	30.25%	22.5%	16%	9.5%
1 link failure	100%	100%	97.27%	74.11%
1 node failure	100%	100%	89.99%	64.72%
2 node failures	93.11%	92.98%	84.05%	58.36%

(a) Single backup in 8 × 8 torus

Muxing degree	mux=1	mux=3	mux=5	mux=6
Spare bandwidth	N/A	30.25%	21.25%	12.88%
1 link failure	N/A	100%	100%	100%
1 node failure	N/A	100%	100%	97.68%
2 node failures	N/A	100%	99.82%	93.28%

(b) Double backups in 8 × 8 torus

Muxing degree	mux=1	mux=3	mux=5	mux=6
Spare bandwidth	33.11%	24.47%	19.69%	17.22%
1 link failure	100%	100%	97.63%	90.39%
1 node failure	100%	99.94%	91.74%	84.08%
2 node failures	89.22%	88.83%	81.82%	75.32%

(c) Single backup in 8 × 8 mesh

Table 1: R_{fast} with same multiplexing degrees

Figure 9. First, the network capacity is reduced by more than 50% for each backup, because a backup channel may be routed over a longer path than the corresponding primary channel.[12] The second backup becomes more expensive than the first backup. Thus, without backup multiplexing, the use of multiple backups will lower the network utilization to an unacceptably low level. Second, the spare bandwidth increases proportionally to the network load regardless of the multiplexing degree. There was no drastic change in the amount of spare bandwidth. Third, with high multiplexing degrees, the overhead of multiple backups becomes close to that of a single backup. See the case of 'mux=6' in Figure 9 (a) and (b). Fourth, in the mesh network, the reduction of spare bandwidth by multiplexing is not as much as in the torus network. This is because the absence of wrapped links in the mesh network makes the primary-channel paths more concentrated on the central region of the network, thus discouraging multiplexing among their backups.

We performed other simulations with inhomogeneous traffic, such as mixed bandwidth requirements or hot-spots in resource reservation. The results indicate that the efficiency of backup multiplexing is relatively insensitive to network traffic conditions, but is more sensitive to network topology — less effective in sparsely-connected networks.

7.2 Degradation of Fault-Tolerance Due to Multiplexing

The next issue to address is the impact of backup multiplexing on fault-tolerance. We assess the fault-tolerance degradation caused by backup multiplexing by simulating three failure models: single link failure, single node failure, and double node failures. Failures were injected into the network after establishing 4032 connections. Each single link failure disabled about 64 primary channels in the torus network, and about 85 primary channels in the mesh network. By injecting each single node failure, about 139 and 276 primary channels were disabled in the torus and mesh network, respectively. Each double node failure caused the disconnection of about 365 and 512 primary channels, respectively. We excluded from consideration the connections whose end nodes fail.

To measure the fault-tolerance level achieved by each backup configuration, we use the fast recovery rate, R_{fast}, as a metric. R_{fast} is the ratio of fast recovery by backup channels to the number of failed primary channels. The $(1 - R_{fast})$ of $\mathcal{D}$-connections, whose primary had failed, required the establishment of new channels for failure recovery. The resultant R_{fast} values are summarized in Table 1, where N/A indicates that the total bandwidth requirement had exceeded the network capacity before establishing all connections.

As expected, the use of a smaller multiplexing degree results in higher fault-tolerance (a larger R_{fast} value). Under the single failure model, R_{fast} solely reflects the impact of backup multiplexing failures, because no connection loses all of its channels due to a single failure. So, 'mux=1' guarantees a perfect recovery coverage from all single failures, and 'mux=3' does from all single link failures. Interestingly, a similar level of fault-tolerance was achievable with significantly less spare resources in the double backup configuration. For example, let's compare the case of single backup with 'mux=3' with the case of double backups with 'mux=6' in the torus network. Using a much smaller spare bandwidth, we achieved comparable R_{fast}, demonstrating the usefulness of multiple backup channels with effective resource sharing. The comparison between double backups with 'mux=6 and a single backup with 'mux=5' more clearly reveals the benefit of the multiple backup configuration.

Considering that the spare bandwidths in Table 1 were measured under the 33 $\sim$ 34% network-load condition, one has to double the spare bandwidth values to estimate the overhead in fully-loaded networks. Thus, in fully-loaded networks (with a 66 $\sim$ 68% network load), 26 – 32% and 34% spare resource overheads were induced in the torus and mesh network, respectively, to achieve around 90% of R_{fast} from single failures. This overhead level can be reduced substantially by employing a more efficient backup routing method. In [HAN97b], we presented a backup routing algorithm which can reduce the spare bandwidth up to 40%, compared to the shortest path routing method.

7.3 Per-Connection Fault-Tolerance Control

So far, we have assumed that all $\mathcal{D}$-connections require the same level of fault-tolerance. We now show how the fault-tolerance level of *each* $\mathcal{D}$-connection is maintained when different connections require different levels of fault-tolerance. We simulated a combination of four types of connections: 1/4 of connections with 'mux=1', 1/4 of connections with 'mux=3', 1/4 of connections with 'mux=5', and the remaining 1/4 of connections with 'mux=6'. The number of backups was the same for all connections. Table 2 shows that the fault-tolerance level of each class of $\mathcal{D}$-connections can be readily controlled, while the overhead remains around the average of all the classes.

7.4 Comparison with Brute-Force Multiplexing

We compare the efficiency of the proposed scheme with that of a simple multiplexing method, called *brute-force multi-*

[12]For example, in a torus network, there are usually two shortest disjoint paths between any two nodes that are more than one hop apart. If the source and destination nodes lie on the same principal axis and the distance between the two is not exactly one half of the torus dimension, there exists only one shortest path.

Spare bandwidth	12.43%			
Muxing degree	mux=1	mux=3	mux=5	mux=6
1 link failure	100%	100%	93.48%	50.43%
1 node failure	100%	99.64%	69.92%	44.14%
2 node failures	93.11%	92.41%	65.88%	39.29%

(a) Single backup in 8 × 8 torus

Spare bandwidth	16.88%			
Muxing degree	mux=1	mux=3	mux=5	mux=6
1 link failure	100%	100%	100%	100%
1 node failure	100%	100%	100%	100%
2 node failures	100%	100%	99.45%	93.67%

(b) Double backups in 8 × 8 torus

Spare bandwidth	17.41%			
Muxing degree	mux=1	mux=3	mux=5	mux=6
1 link failure	100%	100%	97.29%	68%
1 node failure	100%	99.61%	88.15%	52.18%
2 node failures	89.46%	89.04%	78.55%	47.47%

(c) Single backup in 8 × 8 mesh

Table 2: R_{fast} with mixed multiplexing degrees

Spare bandwidth	30.25%	22.5%	16%	9.5%
1 link failure	100%	98.05%	92.19%	76.31%
1 node failure	100%	95.34%	87.98%	68.87%
2 node failures	93.11%	89.82%	82.23%	63.53%

(a) 8 × 8 torus

Spare bandwidth	33.11%	24.47%	19.69%	17.22%
1 link failure	96.18%	89.74%	83.18%	78.18%
1 node failure	96.56%	88.31%	79.49%	72.86%
2 node failures	86.78%	79.62%	71.88%	66.03%

(b) 8 × 8 mesh

Table 3: R_{fast} with brute-force multiplexing

plexing. In the brute-force multiplexing method, the same amount of spare resource is reserved for all links without considering the network status.

First, we applied the brute-force multiplexing to the torus network, with reservation of the same amount of spare resources as the average amount required by our proposed scheme. The comparison between Table 1 (a) and Table 3 (a) shows that the proposed scheme is only marginally better than the brute-force scheme. We attribute this to the homogeneity of the simulated network in terms of network topology, channel traffic model, and the distribution of channel end-nodes. The resource demands for backup activations are therefore evenly distributed throughout the network. In case of a very large multiplexing degree, the proposed scheme's estimation of the spare resource requirement may become less accurate than the brute-force scheme; hence the brute-force scheme results in even higher R_{fast} than the proposed scheme when mux=6.

However, when any sort of inhomogeneity exists, the proposed scheme outperforms the brute-force scheme. The simulation results of the mesh network supports this observation (compare Table 3 (b) with Table 1 (c)). Furthermore, if the channel end-nodes are not evenly distributed or the required bandwidths of all channels are not identical, hotspots (in term of the spare resource demands) occur, and the efficiency of the brute-force scheme degrades significantly unlike the proposed scheme. For the same reason, the proposed scheme outperforms the brute-force scheme in terms of per-connection fault-tolerance control.

8 Related Work

There have been roughly two types of approaches to achieving fault-tolerance in real-time multi-hop networks. The first type is the forward-recovery approach as described in [RAM92, KAO94], where multiple copies of a message are sent simultaneously via disjoint paths to mask failures. A variation of this approach coupled with the error-correction coding scheme can be found in [BAN96]. This approach has an advantage that failures are handled without service disruption, but it is too expensive for certain applications like multimedia networking. If infrequent packet losses due to transient failures are tolerable, the approach to detect and recover from persistent failures is a more attractive and cost-effective alternative. The methods proposed in [BAN93, ZHE92, GRO87, YAN88, BAK91, KAW94, AND94, MUR94] belong to this second type of approach. The proposed scheme also falls into this type.

The method proposed in [BAN93] requires all failures to be broadcast to the entire network. When a source node is notified of the failure of its channel, it tries to establish a new channel from scratch. Since no resource is reserved in advance for the purpose of fault-tolerance, this method has a small overhead in the absence of faults. However, it does not give any guarantee on failure recovery. The channel re-establishment attempt can fail due to resource shortage at that time. Even when there are sufficient resources, the contention among simultaneous recovery attempts for different faulty connections may require several trials to succeed, thus delaying service resumption and increasing network traffic.

In contrast, the method of [ZHE92] provides guaranteed failure recovery under a deterministic failure model (i.e., single failure). In this method, additional resources are reserved in the vicinity of each real-time channel, and the failed components are locally detoured using the resources. Since failures are handled without intervention of source nodes, the recovery latency will be small. However, this method requires reservation of substantial amounts of extra resources, and resource usage becomes inefficient after failure recovery, because channel path-lengths are usually extended by local detouring. Similar approaches in telecommunication networks can be found in [GRO87, YAN88, BAK91].

The work reported in [KAW94, AND94, MUR94] comes closest to our scheme. They proposed VP-restoration methods in ATM networks based on the backup channel concept. The main difference of these from ours is that they are unable to control the fault-tolerance level of each connection (i.e., VP). Another difference is that they assume that a fixed traffic demand (i.e., VP setup requests) is given beforehand and not changed, while we consider the dynamic setup and teardown of channels. Thus, at the network design stage, all channel paths and spare resources are determined together using a computationally very expensive algorithm, to minimize resource overhead while guaranteeing recovery from a certain type of deterministic failures (typically single link failures). Addition or removal of a channel requires recalculation of all channel paths and spare resources. Therefore, these schemes cannot be applied to an environment where short-lived channels are set up and torn down frequently. By contrast, in our scheme, we separated the spare resource allocation problem from the channel routing problem, so that (i) channel path may be selected by any algorithm and (ii) channel establishment may be done in a distributed manner without requiring global knowledge about all channels in the network. We have also presented an *integrated* solution to the problem of channel switching, resource reconfiguration, and control-message transmission, which is not specific to a particular type of network.

9 Conclusion

We have proposed a failure-recovery scheme for dependable real-time communication services in multi-hop networks. The main contributions of this paper are threefold. First, we defined the client interface model for fault-tolerant real-time communication. Second, we devised a mechanism to reduce the fault-tolerance overhead to an acceptably low level. Third, we developed a robust protocol for fast and guaranteed failure recovery. We evaluated the efficiency of the proposed scheme through simulations and showed that with minor degradation of the network's capability of accommodating channels, a desired fault-tolerance QoS level can be achieved.

Acknowledgments

The authors would like to thank Jennifer Rexford, Sugih Jamin, and anonymous reviewers for useful comments on earlier drafts of this paper. The work reported in this paper was supported in part by the National Science Foundation under Grant MIP-9203895, the Office of Naval Research under Grant N00014-94-1-0229, and Mitsubishi Electric Research Laboratory, Cambridge, MA. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[ARA94] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne, "Real-time communication in packet-switched networks," *Proceedings of the IEEE*, vol. 82, pp. 122–139, January 1994.

[MCC96] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," in *Proc. ACM SIGCOMM*, pp. 117–130, August 1996.

[TRI82] K. S. Trivedi, *Probability and Statistic with Reliability, Queuing, and Computer Science Applications.* Prentice-Hall, 1982.

[WHA90] J. Whalen and J. Kenney, "Finding maximal link disjoint paths in a multigraph," in *Proc. IEEE GLOBECOM*, pp. 400–404, 1990.

[SID91] D. Sidhu, R. Nair, and S. Abdallah, "Finding disjoint paths in networks," in *Proc. ACM SIGCOMM*, pp. 43–51, 1991.

[HAN97a] S. Han and K. G. Shin, "Experimental evaluation of failure-detection schemes in real-time communication networks," in *Proc. IEEE FTCS*, 1997.

[ZHA93] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new Resource ReSerVation Protocol," *IEEE Network*, pp. 8–18, September 1993.

[HAN97b] S. Han and K. G. Shin, "Efficient spare-resource allocation for fast restoration of real-time channels from network component failures," *submitted for publication*, 1997.

[RAM92] P. Ramanathan and K. G. Shin, "Delivery of time-critical messages using a multiple copy approach," *ACM Trans. Computer Systems*, vol. 10, pp. 144–166, May 1992.

[KAO94] B. Kao, H. Garcia-Molina, and D. Barbara, "Aggressive transmissions of short messages over redundant paths," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, pp. 102–109, January 1994.

[BAN96] A. Banerjea, "Simulation study of the capacity effects of dispersity routing for fault-tolerant real-time channels," in *Proc. ACM SIGCOMM*, pp. 194–205, August 1996.

[BAN93] A. Banerjea, C. Parris, and D. Ferrari, "Recovering guaranteed performance service connections from single and multiple faults," Tech. Rep. TR-93-066, Computer Science Division, UC Berkeley, 1993.

[ZHE92] Q. Zheng and K. G. Shin, "Fault–tolerant real–time communication in distributed computing systems," in *Proc. IEEE FTCS*, pp. 86 – 93, 1992.

[GRO87] W. Grover, "The selfhealing network: A fast distributed restoration technique for networks using digital crossconnect machines," in *Proc. IEEE GLOBECOM*, pp. 1090–1095, 1987.

[YAN88] C. Yang and S. Hasegawa, "FITNESS: Failure immunization technology for network service survivability," in *Proc. IEEE GLOBECOM*, pp. 1549–1554, 1988.

[BAK91] J. Baker, "A distributed link restoration algorithm with robust preplanning," in *Proc. IEEE GLOBECOM*, pp. 306–311, 1991.

[KAW94] R. Kawamura, K. Sato, and I. Tokizawa, "Self-healing ATM networks based on virtual path concept," *IEEE Journal on Selected Areas in Communications*, vol. 12, pp. 120–127, January 1994.

[AND94] J. Anderson, B. Doshi, S. Dravida, and P. Harshavadhana, "Fast restoration of ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 12, pp. 128–138, January 1994.

[MUR94] K. Murakami and H. Kim, "Near-optimal virtual path routing for survivable ATM networks," in *Proc. IEEE INFOCOM*, pp. 208–215, 1994.