Fast Low-Cost Failure Recovery for Reliable Real-Time Multimedia Communication

Kang G. Shin and Seungjae Han

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

ABSTRACT

Transmission of multimedia data over a packet-switched network typically requires resource reservation to guarantee an acceptable level of performance (e.g., throughput or delay). In this article, we address the problem of how to make such real-time communication reliable. First of all, it is essential to bound the duration of service disruption caused by failures to a reasonably small value. Considering the large volume of multimedia data, minimizing the fault-tolerance overhead is also important. Furthermore, as more applications with different dependability requirements share the same network, the level of dependability for a given application should be "customizable," depending on the criticality of the application. We first survey the existing approaches, and then present our scheme which is developed in accordance with three design goals — fast failure recovery, low fault-tolerance overhead, and per-connection reliability guarantee. Our scheme provides an integrated solution covering such issues as connection establishment, failure detection, run-time failure recovery, and resource reconfiguration.

Index Terms — Real-time communication, fast failure recovery, dependability guarantee, backup channel.

1 Real-time Multimedia Communication

Real-time transport of continuous media (i.e., video and audio) has traditionally been achieved by circuit switching in telephone services or by broadcasting over shared media in television services. However, it is difficult to realize real-time continuous multimedia applications on packet-switched computer networks, since the end-to-end packet delay and throughput of a media stream are inherently non-deterministic. Such end-to-end performance necessary to achieve the required functionality of these applications is often called end-to-end *Quality-of-Service* (QoS). Today's representative computer network, Internet, also lacks QoS support for continuous media applications; the window-based flow control is unsuitable for traffic with end-to-end timing constraints. Nevertheless, several multimedia applications have already been deployed on Internet using such protocols as RTP [1], XTP [2], and IP multicast. However, these protocols do not meet the true multimedia requirements because they only support a best-effort service model. The Next-Generation Internet (NGI) is expected to provide new services that meet the diverse QoS requirements of various emerging multimedia applications.

In response to this growing demand for real-time communication services, considerable efforts have been made in recent years and numerous QoS service models have been developed ranging from the static CBR (Constant Bit Rate) service that resembles the telephony service to the 'controlled-load service' that mimics the best-effort service in unloaded networks [3, 4]. Unlike traditional datagram services in which average performance is of prime interest, guaranteeing QoS is the key requirement of the real-time communication service. To achieve QoS guarantees, most of the real-time communication schemes rely on some form of resource reservation and admission control, while each differs in QoS parameters and/or the firmness of QoS guarantees. That is, they share three common properties: *QoS-contracted*, *connection-oriented*, and *reservation-based*. A contract between a client and the network is established before the client's messages are actually transferred. To this end, the client must first specify his input traffic behavior and required QoS. The network then 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 client's QoS requirement, the request is rejected. The client's data messages are transported only via the selected path with the resources reserved, and this virtual circuit is often called a *real-time channel*.

2 Network Dependability

Primitive real-time communication services will soon be available for such multimedia applications as Internet phone, WWW, digital libraries, etc. For example, the controlled-load service is considered as a possible candidate. On the other hand, the increase of network connectivity and link capacity will expand the application domain of real-time multimedia communication to include business- or mission-critical applications, such as remote medical service, collaborative scientific research, business net-meeting, real-time electronic commerce, or even remote battle-field command/control. Such critical applications require *both* dependable and timely communication services. Suppose, for example, there is a very important video conference and unanticipated 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. In this example, the connection availability defined as the probability of a connection being available at any given time is a key dependability QoS.

Network failures can cause even a larger-scale social disaster. Catastrophic social consequences of network failures have actually been witnessed in recent breakdowns of the US telecommunication network. For instance, the fire at an unmanned tall office in Illinois caused 3.5 million telephone calls to be blocked in 1988. Emergency 911 calls went unanswered, on-line business transactions were stopped, and flights were delayed because of air traffic controller failures. Hospital operations were affected and drug stores could not process prescriptions. Even banks had to be closed for security reasons due to disabled alarm systems [5]. In 1990's, several similar accidents have been reported for various reasons like damage of a fiber cable caused by construction, earthquakes, outage of switching systems, or network overloading. Although network failures rarely occur, the consequence of mis-handling failures could be devastating, thus making network reliability a major concern.

The current Internet with datagram services has successfully dealt with two types of network failures: *transient* and *persistent* failures. A typical example of transient failures is temporary packet losses due to either network congestion or data corruption. Persistent failures include breakdown or crash of network components. Transport protocols like TCP can handle transient loss of packets by acknowledgment and retransmission, and the connection-less IP protocol deals with persistent failures by routing packets around faulty network components. However, retransmission is unlikely to be useful for real-time communication, because there is usually not enough time to detect and retransmit a lost real-time message before its deadline expires. Instead, for real-time communication, forward error-correction (FEC) techniques should be used if no data loss is acceptable. The main drawback of FEC is its high overhead. We also face a serious difficulty in tolerating persistent failures for real-time communication, because a QoS guarantee is realized by reserving resources on a fixed path and transporting real-time messages only via the path. Hence, a real-time message, unlike datagram messages, cannot be detoured around faulty components on the fly.

The prevalence of optical fibers affects network dependability in two different ways. First, the probability of transmission errors in optical links becomes negligible; the error rate is dropped from 10^{-5} per bit in the 56 kbps links of initial ARPANET to below 10^{-10} per bit in optical links [6]. The chance of packet loss due to transmission errors is very rare, and most packet losses are attributed to congestion-control problems. As a result, for real-time communication, tolerating transient failures has become relatively less important because congestion-induced packet losses can be avoided by resource reservation. Furthermore, occasional loss of messages is tolerable in many multimedia applications. By contrast, the deployment of optical fibers exacerbates the difficulty in tolerating persistent failures, because more connections will be running through each large-capacity link, and thus, even a single link failure can result in loss of a large number of connections. Unless the network is carefully designed to restore and then deliver the large amount of traffic lost due to failures, the increase of link capacity will seriously threaten the network dependability. Not only link failures but also node failures are getting more difficult to deal with. Usually, Internet routers are not designed to meet such a stringent reliability goal as telephone network switches (e.g., AT&T 5ESS). Higher-performance routers in the future Internet will become even harder to provide high reliability, due mainly to their complex software. Moreover, computer networks are more vulnerable to vandalism like virus or hacking than the telephone network which has a "closed" architecture.

Considering the criticality of network dependability and the increasing threat of network failures, the development of effective mechanisms to cope with network failures is a *must*.

3 Desirable Features

To design a fault-tolerant service, one must first define the model of failures to be tolerated. Some applications can tolerate slow failure recovery but require reliable (correct) delivery of all messages even if it takes a long time. Examples of such applications are electronic mail or file transfer. For these applications, message-level failure handling is necessary. Thus, the receiver informs the sender of the reception of each message (or a group of messages), so that the sender can detect delivery failures and retransmit lost messages. Some other applications require fast failure recovery but the data loss during failure recovery can be tolerated. Real-time multimedia applications fall into this category, since they do not require such strict reliability as 'no message loss at all.' For example, loss of a couple of frames in video/voice data streams may be acceptable. Therefore, the fault-tolerant service for multimedia applications should be designed with a different reliability goal from that of conventional non-real-time applications.

In this article, we assume that (infrequent) transient packet losses are acceptable to applications, or are dealt with by other techniques like FEC, and focus on how to effectively handle "persistent" or "permanent" failures of network components, e.g., *crash failures*. Instead of directly modeling component failures, we define and use a new failure semantic, *channel failure*. A real-time channel is said to have 'failed,' if the rate of correct[1] message delivery within a certain time interval is below a threshold specified by the application; i.e., the granularity of failures is 'persistent disruption of channels' instead of 'individual message loss.' The same error rate may be acceptable to some applications while it may not to other applications.

There are five criteria that characterize a good solution to this problem:

- **Per-connection dependability guarantee:** Each connection may request a different level of fault-tolerance depending on its criticality. The network should provide guarantees on dependability for each connection, so that successful recovery is guaranteed as long as failure occurrences do not exceed the fault-tolerance capability of the connection.

- **Fast (time-bounded) failure recovery:** The service-disruption time of a connection caused by failures should be bounded to a reasonably small value.

- **Small fault-tolerance overhead:** The additional resource overhead required for fault-tolerance should be acceptably low.

- **Robust failure handling:** Failures should always be handled robustly even though failure occurrences may exceed the assumed failure hypothesis. By 'robustly,' we mean that the QoS of nonfaulty real-time channels is not affected, and as many faulty real-time channels as possible are recovered.

- **Interoperability/scalability:** The failure recovery scheme must be interoperable with various existing and future real-time channel protocols. Also, it should scale well in a dy-

[1] In terms of both content and timing.

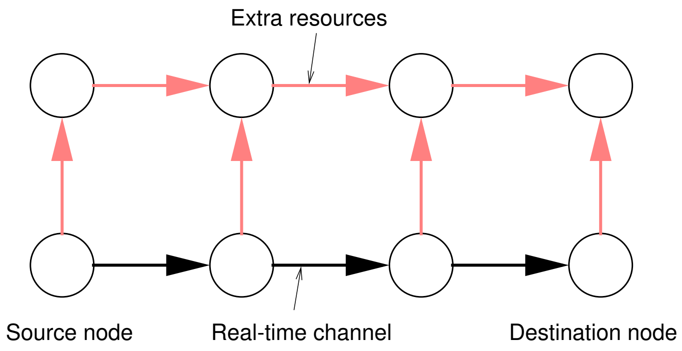


Figure 1: Example of a SFI channel

namic environment where (short-lived) connections are set up and torn down frequently and the distributions of connection setup requests and connection holding time are unknown.

4 Existing Approaches

Before presenting our approach, we will first survey some notable previous work on dependable real-time communication. Particularly, the schemes developed for computer and telephone networks are reviewed with comparative discussions against our approach.

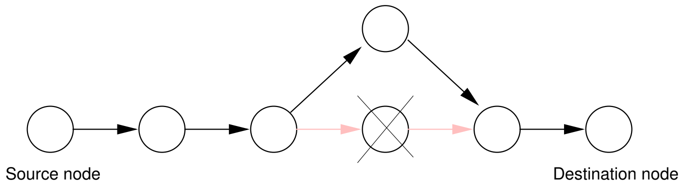
4.1 Packet-switched Computer Networks

There have been roughly three types of approaches to the problem of achieving dependable real-time communication in packet-switched computer networks.

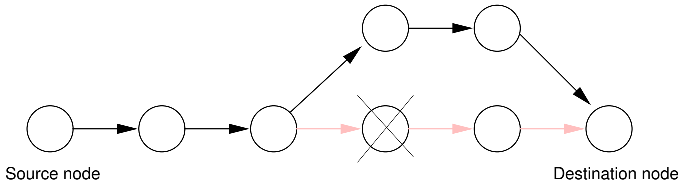
First, the simplest way of recovering a real-time channel from a network-component failure is to establish a new real-time channel which exclude the failed component. This *reactive* method is studied in [7]. This scheme relies on the broadcast of all component failures to the entire network, so that all hosts can maintain a consistent view on the current network topology. When a source node recognizes the failure of its channel from this broadcast, it tries to establish a new channel to replace the disabled channel. Since no provisions are made *a priori* for fault-tolerance, this method causes no fault-tolerance overhead in the absence of failures. However, it does not give any guarantee on failure recovery. The channel re-establishment attempt can fail due to resource shortage at that particular 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. To regulate simultaneous recovery attempts, random delays can be introduced before starting each recovery operation.

The second approach is failure-masking in which multiple copies of a message are sent simultaneously via disjoint paths [8]. This method attempts to achieve both timely and reliable delivery at the same time. Thus, by transmitting multiple copies over disjoint paths for a message, the chance that at least one copy is delivered within its deadline increases and the effects of possible failures are masked. This approach has an advantage that both persistent and transient failures are handled without service disruption, but it is very expensive due to the additional resource consumption for transmitting multiple copies of the same message. To enhance resource efficiency, [9] presents a scheme which combines error-coding with multiple-copy transmission. In this method, instead of transmitting multiple copies of an entire message, each message is broken into equal-size sub-messages which are then transmitted over disjoint paths. In addition, some redundant information is transmitted over separate paths for FEC. When some sub-messages are lost due to failures, the original message can be reconstructed using the redundant information. This method allows for making a tradeoff between resource overhead and fault-tolerance capability by selecting the number of redundant information channels. However, this method still involves the additional resource consumption for FEC, which is not necessary if the underlying multimedia applications can tolerate infrequent transient data loss. Another practical problem is that the connectivity of current (or near-future) network topologies may not provide a sufficient number of disjoint paths between any two network nodes, which may hinder the flexibility of this method.

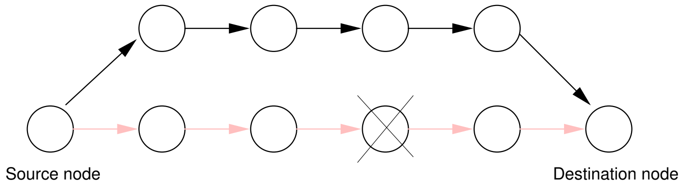
The third approach comes in-between the above two approaches in terms of fault-tolerance overhead. In this approach, cold-standby resources are reserved for the fault-tolerance purpose. The SFI (Single Failure Immune) scheme [10] took this approach to provide guaranteed failure recovery under a single failure model. In this scheme, additional resources are reserved in the vicinity of each real-time channel at the time of channel establishment, and when a failure occurs the failed component is detoured by altering the channel path using the reserved resources. Figure 1 illustrates the setup of a SFI channel. The advantage of this cold-standby approach is that, though additional resources need to be reserved, the resources reserved for fault-tolerance can be utilized by best-effort traffic in the absence of failures. Another cold-standby method is presented in [11]. In this method, cold-standby reservation is combined with multiple-channel transmission. It differs from the multiple-channel FEC method in that extra sub-channels remain as cold-standby in absence of failures; thus, FEC is not provided. The cold-standby extra channels are activated to replace the original sub-message channels disabled by failures.



(a) Local rerouting



(b) Local-to-end rerouting



(c) End-to-end rerouting

Figure 2: Three rerouting strategies

4.2 Telephone Networks

In old telephone networks, two telephones were connected by a true electric circuit through electro-mechanical or pure electric exchanges. Even after the introduction of digital transmission hierarchies like the SDH (Synchronous Digital Hierarchy) and SONET (Synchronous Optical Network) transmission standard, the 64 kbps circuit-switching paradigm continued as a main technology. However, the transition from circuit- to cell-switching (i.e., ATM network) has completely changed the problems and solutions of telephone networks. Now, telephone networks are very close to computer networks. A modern switching node in telephone networks is almost a general-purpose computer equipped with high fault-tolerance capability and powerful I/O capability. A mesh-like network topology is being used instead of a fully-connected topology. Techniques for telephone services have resemblance to those for real-time communication services in packet-switched computer networks, in that both services rely on similar principles such as dedicated resources and static routing. Therefore, the dependability techniques of telephone networks are worth a close look. While the telephone network survivability is accounted for at various levels, here we cover only the network-layer operation, which is most relevant to this article.

Essentially, when a telephone connection is broken, the connection is rerouted by detouring the failure point. Failure recovery should be fast so that users (or applications) may hardly notice the service disruption caused by the failure. Even more important is to ensure the success of failure recovery itself. If there are not enough resources available for rerouting all affected connections, some of them should be dropped. To avoid resource shortage during failure recovery, '*spare resources*' (i.e., cold-standby resources) are reserved in advance. The amount of spare resources to be allocated is an important design issue, and is closely related to the problem of selecting the rerouting paths of failed connections. For rerouting, there are three strategies: *local-rerouting*, *local-to-end rerouting*, and *end-to-end rerouting*. Each of these strategies is illustrated in Figure 2.

The local-rerouting strategy, also called a *span-restoration* method, has usually been used in STM (Synchronous Transfer Mode) networks. In most of the work on this strategy [12, 13], a 'maximum flow' model is used to find the (semi-) optimal placement of spare resources under a deterministic failure hypothesis — typically, a single-link failure model. A drawback of the local-rerouting approach is that resource usage becomes inefficient after failure recovery, because channel paths tend to be lengthened by local detouring. According to [14], end-to-end rerouting is the best with respect to resource efficiency, local-to-end rerouting is the second, and local rerouting is the worst.

The *end-to-end rerouting* strategy, also called a *path-restoration* method, has been studied mainly in the context of ATM networks. Essentially, this strategy requires the intervention of the end-points of each failed connection. There are two variations in this strategy, depending on whether the failure recovery paths are pre-computed before failure occurrence, or the recovery paths are determined after failures actually do occur. There are several differences between these two methods. In the former, the pre-routed recovery paths should be disjoint with the corresponding original connection paths, where in the latter the recovery paths can use the healthy components of their original connection paths. The higher flexibility in routing the recovery paths implies the potential of higher resource efficiency of the latter. However, the former has an advantage over the latter in terms of dependability guarantees. In the former, since the network can reserve the spare resources necessary for the recovery connections whose paths are already decided, there will be no conflict/contention for spare resources between recovery attempts upon failures. By contrast, in the latter, when failures occur, each disabled connection will try to establish a new connection by "claiming" the shared spare resources. Therefore, resource contention similar to the reactive method can happen, and some connections may need to try several recovery paths before they succeed. Some of recent efforts on the former method

	Resource overhead	Recovery delay	Recovery guarantee
Reactive	No	Long	No
SFI	High	Shorter	Deterministic
Multi-copy	Very high	No	Flexible
Span-restoration	Low	Shorter	Deterministic
Path-restoration	Lower	Short	Deterministic
Our approach	Lower	Short	Flexible

Table 1: Comparison of existing approaches

can be found in [14–16], and an example of the latter method is [17]. Essentially, they try to optimally route recovery paths by minimizing the spare resource reservation, while guaranteeing successful recovery under a deterministic failure model.

4.3 Comparison

Basically, our approach uses end-to-end rerouting with pre-computed recovery paths. We set up one or more *backup channels* in advance in addition to each *primary channel*; that is, each dependable real-time ($\mathcal{D}$-) connection consists of one primary channel and backup channels. Upon failure of a primary channel, one of its backups is promoted to a new primary channel.

In Table 1, existing approaches and our approach are compared with each other in terms of resource overhead, recovery delay, and recovery guarantee. The SFI method is similar to the span-restoration method in many aspects, except that it induces higher overhead, because, in the SFI method, the entire spare resources required by each connection is reserved unlike the span-restoration method which optimizes the allocation of spare resources. Both methods will have a shorter recovery delay than the end-to-end rerouting methods, because failures are handled locally without intervention of end-nodes. The recovery delay of our approach is smaller than that of the reactive method, because in our approach a backup channel is established before failures actually occur and the network can use it immediately upon the failure of its primary channel, without the time-consuming channel (re)establishment process.[2]

[2]The time required to establish a real-time channel is relatively large and can be unbounded even without contention, since channel-establishment requests are usually sent as best-effort messages and the admission test is required at each node on the channel path.

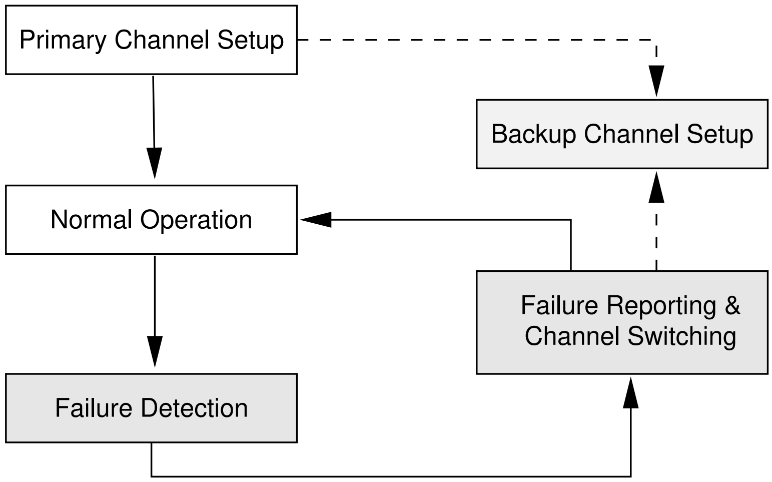


Figure 3: Overview of our failure-recovery scenario

The path-restoration method for ATM networks comes closet to our approach. But, there exist three main differences between the two. First, the path-restoration method is unable to control the fault-tolerance level of each connection, and all connections are treated equally under the same failure model. By contrast, our approach allows for per-connection fault-tolerance control, so that more critical connections will get higher levels of fault-tolerance. Second, in the path-restoration method, all channel paths and spare resources are determined simultaneously, and hence, addition or removal of a channel requires recalculation of all channel paths and spare resources. It is assumed that the connection demands (i.e., VP setup requests) are known at the time of network design and change very rarely.[3] Therefore, this method cannot be applied to an environment where short-lived channels are set up and torn down frequently. By contrast, out approach needs only the information which can be easily obtained at run-time. It is possible because we separate the spare-resource allocation problem from the channel routing problem, so that (i) a backup path may be selected by any algorithm and (ii) spare-resource allocation may be done with given routing results. Third, although the control of recovery procedures might be distributed, centralized, or a hybrid of the two, calculation/assignment of spare resources is usually centralized in the path-restoration method. In our approach, both run-time failure recovery and spare resource allocation are done in a fully distributed manner.

Figure 3 gives an overview of our failure-recovery scenario. The key steps in our approach are: (i) backup channel establishment, (ii) failure detection, (iii) failure reporting and channel switching, (iv) resource reconfiguration. The rest of this article will outline each of these steps.

[3]In telephone ATM networks, each call setup is handled at the VC level without requiring a new VP to be set up.

5 Connection Establishment

A backup channel does not carry any data until it is activated, so that it does not consume bandwidth in a normal situation. However, a backup channel is not free, as it requires the same amount of resources to be reserved as its primary channel, in order to provide the same QoS as its primary upon its activation. In a normal situation, spare resources can be used by non-real-time traffic, but they cannot be used to set up other real-time channels. This is because if a real-time channel is established by using spare resources which are reserved for other backups, its QoS guarantee may be violated when spare resources are claimed for failure recovery. 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 (because the backup is likely to run over a longer path). Thus, raw backup channels are too expensive to be useful for multimedia networking.

To alleviate this problem, we have developed a resource-sharing technique, called *backup multiplexing*. Its basic idea is that on each link, we reserve only a very small fraction of link resources needed for all backup channels going through the link. In this article, we consider only link-bandwidth for simplicity, but other resources like buffer and CPU can be treated similarly.

With backup multiplexing, backup channels are overbooked via a *meta-admission test*, in which some existing backup channels are not accounted for in the admission test of a new backup channel. It is, in essence, equivalent to resource sharing between the new backup and those backups unaccounted for. Deciding which backup channels will not be accounted for in the admission test of a backup channel is a crucial problem. Our strategy is to multiplex those backups which are less likely to be activated simultaneously. The probability of simultaneously activating two different $\mathcal{D}$-connections' backups, B_i and B_j, is equal to — actually, bounded by — the probability of simultaneous failures of their respective primary channels. Let's denote this probability by $\mathcal{S}(B_i, B_j)$. Based on this probability, the set of backups to be multiplexed together is determined for each backup on a link, i.e., hop-by-hop multiplexing. B_i and B_j are multiplexed if $\mathcal{S}(B_i, B_j)$ is smaller than a certain threshold ν, called *multiplexing degree*, which is specific to each backup. More accurately, if $\mathcal{S}(B_i, B_j) < \nu_i$, B_j can be multiplexed with B_i. Each backup can use a different ν to determine the set of backups to be multiplexed. The smaller ν of a backup, the higher fault-tolerance will result, since less backups will be multiplexed with it. This way, more important connections can have higher fault-tolerance (e.g., tolerating harsher failures). However, we require each backup to have the same multiplexing degree on all of its links for easier management.

Figure 4 describes an algorithm that calculates the amount of spare resources (s_ℓ) at link ℓ

loop for each backup channel B_k on link ℓ

if $\mathcal{S}(B_k, B_i) \geq \nu_i$ and $\nu_k \leq \nu_i$ **then**

$\Pi_{B_i,\ell} \longleftarrow \{\Pi_{B_i,\ell} + B_k\}$

endif

endloop

$s_\ell \longleftarrow max\{(\sum_k^{\Pi_{B_j,\ell}} r_k) + r_j\}, \forall B_j \in \ell$

Figure 4: Backup multiplexing algorithm

when a new backup B_i is established on ℓ. Let $\Pi_{B_i,\ell} = \{B_\alpha, B_\beta, \ldots\}$ denote the set of backups which are *not* multiplexed with B_i on ℓ, and r_k is the resource requirement of B_k. After calculating all $\Pi_{B_i,\ell}$, the highest resource requirement among all sets of $\{\Pi_{B_i,\ell} + B_i\}$ is chosen as s_ℓ. To construct $\Pi_{B_i,\ell}$, we consider only backups with no greater multiplexing degrees than that of B_i. This is to avoid the risk of overestimating the spare-resource requirement by accounting for all backups regardless of their multiplexing degrees.

In establishing a $\mathcal{D}$-connection, we consider two key dependability-QoS parameters (P_r, Γ), P_r is the probability of fast failure recovery, and Γ is the estimated failure-recovery delay. In other words, the probability that a $\mathcal{D}$-connection will suffer a service disruption longer than Γ is not greater than P_r. As fast failure recovery relies on the availability of backups, P_r of a $\mathcal{D}$-connection increases with the number of its backups. Backup multiplexing also affects P_r in the following way. It reduces the amount of spare resources by reserving less resources than the sum of resources needed by individual backups, but creates a possibility of spare resource exhaustion. Thus, some backups cannot be activated because other activated backups have already taken all spare resources. In such a case, a "*multiplexing failure*" is said to occur. We account for the probability that a backup suffers a multiplexing failure in the calculation of P_r. See [18] for a detailed account of calculating P_r.

Thus far, we explained the procedure of allocating spare resources, assuming that backup paths are given. How to route backups greatly influences the resource efficiency, i.e., the amount of spare resources to achieve a certain P_r. Our study shows that traditional routing algorithms such as minimum-hop routing or maximum load-balanced routing are less effective in backup routing, as compared to the routing methods which capitalize the characteristics of backup multiplexing.

6 Failure Detection

Effective failure detection with high coverage and low latency is essential for fast failure recovery. Instead of adopting expensive failure-detection techniques (e.g., hardware duplication/comparison for telephone network switches [19]), we use behavior-based detection techniques do not require any special hardware support and hence, can be used in *any* network. They are the end-to-end detection and neighbor detection methods.

The end-to-end detection method involves both the source and destination nodes of a real-time channel. The source node regularly injects "channel heartbeats" into the channel message stream. A channel heartbeat is a sort of real-time message, and the intermediate nodes on a channel do not discriminate channel heartbeats from data messages. Each channel heartbeat contains the sequence number of the latest data message. In this way, the destination node can monitor the number of data messages lost. If the message-loss rate of the channel exceeds the channel's threshold, the destination node declares that the channel has failed. This method will uncover *all* channel failures, i.e., 100% failure detection coverage. But, its detection latency is tightly coupled with the application-specific failure threshold, so that applications with large failure thresholds will result in long detection latencies.

The neighbor detection method resembles the gateway failure-detection protocols in the Internet. Adjacent nodes periodically exchange node heartbeats ("I am alive"). If a node does not receive heartbeats from one of its neighbors for a certain period, it declares all the channels going through the silent neighbor as failed. This scheme has a much weaker dependency on the channel-specific characteristics than the end-to-end scheme, because it monitors the behavior of a neighbor node, rather than that of a particular real-time channel. If a node crashes quickly after failures occur, one can achieve very small detection latencies by this scheme. However, there is a possibility that a node does not correctly transmit data messages, but still generates node heartbeats for a while. In such a case, the neighbor scheme may miss the failures of some channels or may detect them later than the end-to-end scheme.

We experimentally evaluated the effectiveness of the two failure-detection schemes. The experimental results on a laboratory testbed, which was designed without any particular consideration for fault-tolerance, have indicated that the neighbor scheme detects a significant portion of failures very quickly, while a long latency occasionally results. (Detailed experimental results are reported in [20].)

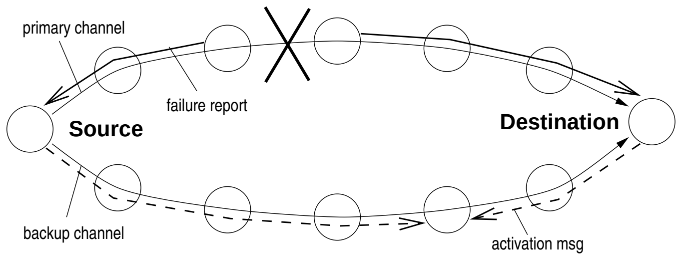


Figure 5: An example of failure reporting and backup activation

7 Failure Reporting & Channel Switching

If the node which detects the failure of a channel is different from the node which is responsible for channel switching, the detected failure should reported to the latter node. There are three important issues in reporting failures. First, who needs to receive failure reports ? Second, which path will be used for reporting failures ? Third, what information needs to be carried in a failure report ? Our approach to these issues is: (i) failure reports are sent from the failure-detecting nodes 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 failure report about the same channel had already been received by, or passed through, the node. 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. If the spare resources at a link are exhausted by the activation of backups, the remaining backup channels on the link cannot function as standby channels, i.e., multiplexing failures. Multiplexing failures should also be reported the same as component failures.

When an end-node of a $\mathcal{D}$-connection receives a failure report on its primary channel, it selects one of its healthy backups[4] and sends an '*activation message*' along the path of the selected backup. (Both end-nodes may start backup activation simultaneously.) 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. Figure 5 illustrates the channel switching procedure described above. If a failure is detected by the channel's destination node (e.g., with the end-to-end method), there will be no explicit failure reporting, and instead, the destination node will immediately start the backup activation procedure. To minimize service

[4]The health of backups should also be monitored.

	Case 1	Case 2	Case 3
primary channel	×	○	×
backup channel	×	×	○

Table 2: Cases requiring resource reconfiguration

disruption, data transfer through the new primary channel will be resumed immediately after the source node sends an activation message or receives an activation message generated by the destination node. 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.

The transmission of failure reports and activation messages is time-critical, because their delays directly affect the service-disruption time. The delay of such messages is unpredictable if they were transported as best-effort messages. Assigning the highest priority to such messages is not a good solution either, as it may affect the QoS of regular real-time traffic. Suppose there are malicious nodes or a large number of coincident failures. In such cases, flooding urgent messages can paralyze the whole (or part of the) network. To achieve delay-bounded and robust transmission of time-critical control messages, we transmit them over special-purpose real-time channels, called the *real-time control channel* (RCC). When the network is initialized, a pair of RCCs, one in each direction, is established on every link of the network. If the capacity of the RCC on each link is large enough to accommodate all time-critical control messages on the link, the timely delivery of such messages can be guaranteed.

8 Resource Reconfiguration

In a normal situation, the dependability QoS of a $\mathcal{D}$-connection is maintained by limiting the admission of new connections not to impair the QoS of existing connections. Upon occurrence of a failure, more explicit actions (i.e., resource reconfiguration) need to be taken to preserve the QoS of the connections which are directly or indirectly affected by the failure. Specifically, there are three cases which require resource reconfiguration for QoS maintenance as shown in Table 2 where $\times$ denotes failure and $\bigcirc$ non-failure (thus healthy).

In Case 1, both primary and backup channels will be re-established. In Case 2, the faulty backup will be torn down and a new backup will be established to maintain the dependability QoS of the injured connection. The network should tear down the old backup before a new backup is established, so that the new backup can be routed using the healthy components on the old backup path, if necessary. In Case 3, the healthy backup will be promoted to a new primary channel, where the faulty primary will be torn down. After channel switching, a new backup is necessary since the original backup has been activated, thus ceasing its backup role.

Even when a connection is not directly inflicted by failures, its dependability QoS can be affected by the failure recovery for other connections. It is because spare resources are shared by multiple backups and activation of a backup will reduce the spare resources on its path, and as a result, the remaining backups on this path may not receive their original QoS.[5] At such links, more spare resources have to be allocated to maintain the same QoS for the remaining backups. Here the network has to take care of a situation when there are not enough resources available at a link to match the need of additional spare resources. The network can resolve such situations by moving some of the remaining backups to different paths or by QoS degradation. In this article, we will not address this issue any further.

9 Performance Analysis

We evaluated the resource overhead and fault-tolerance capability of our scheme through simulations. The simulated networks were an 8×8 torus (wrapped mesh) and an 8×8 mesh. To obtain a similar total capacity for both networks, we set the link capacity of the torus to 200 Mbps and set that of the mesh to 300 Mbps. At each simulation run, a total of 4032 connections were established so that the source and destination of each connection were evenly distributed. Channels of each $\mathcal{D}$-connection were routed disjointly with the shortest-path routing. For simplicity, we assumed that all channels used the same traffic model so each channel required 1 Mbps of bandwidth on each link of its path. We also assumed that the same multiplexing degree was used for all backup channels. Three failure models (i.e., single link failure, single node failure, and double node failures) were simulated. As a metric of the fault-tolerance level achieved by each backup configuration, the ratio of fast recovery to the number of failed primary channels was used. For instance, 90% of fast recovery ratio means that 90% of the $\mathcal{D}$-connections, whose primary failed, were recovered by using their backup channels.

The simulation results are summarized in Table 3. The fast recovery ratio given in Table 3 is

[5] In the worst case, the remaining backups can be subject to the exhaustion of spare resources.

	$\nu = 0$	$\nu = \lambda$	$\nu = 3\lambda$	$\nu = 5\lambda$	$\nu = 6\lambda$
Spare bandwidth	35%	30.25%	22.5%	16%	9.5%
1 link failure	100%	100%	100%	97.27%	74.11%
1 node failure	100%	100%	100%	89.99%	64.72%
2 node failures	93.11%	93.11%	92.98%	84.05%	58.36%

(a) Single backup in 8 $\times$ 8 torus

	$\nu = 0$	$\nu = \lambda$	$\nu = 3\lambda$	$\nu = 5\lambda$	$\nu = 6\lambda$
Spare bandwidth	N/A	N/A	30.25%	21.25%	12.88%
1 link failure	N/A	N/A	100%	100%	100%
1 node failure	N/A	N/A	100%	100%	97.68%
2 node failures	N/A	N/A	100%	99.82%	93.28%

(b) Double backups in 8 $\times$ 8 torus

	$\nu = 0$	$\nu = \lambda$	$\nu = 3\lambda$	$\nu = 5\lambda$	$\nu = 6\lambda$
Spare bandwidth	35.47%	33.11%	24.47%	19.69%	17.22%
1 link failure	100%	100%	100%	97.63%	90.39%
1 node failure	100%	100%	99.94%	91.74%	84.08%
2 node failures	89.11%	89.22%	88.83%	81.82%	75.32%

(c) Single backup in 8 $\times$ 8 mesh

Table 3: Simulation results (λ = component failure rate)

the average value collected from all possible failure cases under the corresponding failure model. The spare bandwidth value is the average of all links. N/A indicates that the total bandwidth requirement had exceeded the network capacity before establishing all connections.

In general, with far less than one half of resource overhead as compared to that before multiplexing[6], more than 90% of fault-tolerance capability could be preserved by backup multiplexing in the single backup configuration. As expected, the use of a smaller multiplexing degree results in higher fault-tolerance with an exception between '$\nu = 0$' and '$\nu = \lambda$' at the bottom

[6] '$\nu = 0$' has the same effect as disabling backup multiplexing, since no backup will be multiplexed with each other.

line of Table 3 (c), where the fault-tolerance level was increased after backup multiplexing. It is due to the impact of backup multiplexing on the channel route selection. The multiplexing efficiency was even further improved by using the double backup configuration. Thus, a similar level of fault-tolerance was achievable with significantly less spare resources in the double backup configuration. For example, compare the case of single backup with '$\nu = 3\lambda$' with the case of double backups with '$\nu = 6\lambda$' in the torus network. Another interesting observation is that the multiplexing efficiency is affected by the network topology. In the mesh network, the reduction of spare resources 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.

10 Concluding Remarks

The main focus of this article is twofold. First, we surveyed existing approaches to dependable multimedia communication and discussed their pros and cons. Second, we presented a new scheme which selectively adopts only the merits of existing approaches without their shortcomings. Thus, this scheme allows the desired level of fault-tolerance to be achieved for each connection at an acceptably low level of resource overhead.

This 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 be broadcast. Control messages are sent only over those paths of channels affected by failures. 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, by making a backup channel-establishment message carry the path information of its primary channel. Since our scheme is designed without any assumption for a particular real-time communication scheme, it can be placed on top of any existing (possibly independently developed) real-time channel protocols.

References

[1] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," Technical Report Internet RFC 1889, February 1996.

[2] *Xpress Transfer Protocol Specification*, XTP Forum, revision 4.0 edition, March 1995.

[3] 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, no. 1, pp. 122–139, January 1994.

[4] D. Ferrari, "Multimedia network protocols: where are we?," *Multimedia Systems Journal*, vol. 4, pp. 299–304, 1996.

[5] J. McDonald, "Public network integrity – avoding a crisis in trust," *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 1, pp. 5–12, January 1994.

[6] A. Tanenbaum, *Computer Networks, 3rd ed.*, Prentice-Hall, Englewood Cliffs, New Jersey, 1996.

[7] A. Banerjea, C. Parris, and D. Ferrari, "Recovering guaranteed performance service connections from single and multiple faults," Technical Report TR-93-066, UC Berkeley, 1993.

[8] P. Ramanathan and K. G. Shin, "Delivery of time-critical messages using a multiple copy approach," *ACM Trans. Computer Systems*, vol. 10, no. 2, pp. 144–166, May 1992.

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

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

[11] A. Banerjea, *Fault Management for Realtime Networks*, PhD thesis, University of California at Berkeley, 1994.

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

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

[14] J. Anderson, B. Doshi, S. Dravida, and P. Harshavadhana, "Fast restoration of ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 1, pp. 128–138, January 1994.

[15] 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, no. 1, pp. 120–127, January 1994.

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

[17] R. Iraschko, M. MacGregor, and W. Grover, "Optimal capacity placement for path restoration in mesh survivable networks," in *Proc. IEEE ICC*, pp. 1568–1574, 1996.

[18] S. Han and K. G. Shin, "Fast restoration of real-time communication service from component failures in multi-hop networks," in *Proc. ACM SIGCOMM*, pp. 77–88, 1997.

[19] W. N. Toy, "Fault-tolerant design of AT&T telephone switching system processors," in *Reliable Computer Systems: Design and Evaluation*, pp. 533–574. Digital Press, 1992.

[20] S. Han and K. G. Shin, "Experimental evaluation of failure-detection schemes in real-time communication networks," in *Proc. IEEE FTCS*, pp. 122–131, 1997.

Biography

Kang G. Shin is Professor and Director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan.

He has authored/coauthored about 600 technical papers and numerous book chapters in the areas of distributed real-time computing and control, computer networking, fault-tolerant computing, and intelligent manufacturing. He has co-authored (jointly with C. M. Krishna) a textbook "Real-Time Systems," McGraw Hill, 1997. In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award, and in 1989, Research Excellence Award from The University of Michigan. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are investigating various issues related to real-time and fault-tolerant computing.

His current research focuses on Quality of Service (QoS) sensitive computing and networking with emphases on timeliness and dependability. He has also been applying the basic research results to telecommunication and multimedia systems, intelligent transportation systems, embedded systems, and manufacturing applications.

He received the B.S. degree in Electronics Engineering from Seoul National University, Seoul, Korea in 1970, and both the M.S. and Ph.D degrees in Electrical Engineering from Cornell University, Ithaca, New York in 1976 and 1978, respectively.

Seungjae Han is a post-doctoral research fellow of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, the University of Michigan, Ann Arbor, Michigan. He received both the B.S. and M.S. degrees in Computer Engineering from Seoul National University, Seoul, Korea, in 1989 and 1991, respectively, and the Ph.D degree in Computer Science and Engineering from the University of Michigan in 1998. His research interests include computer networks, fault-tolerant systems, and real-time systems.