Chapter 1: Introduction

1.1. The realtime channel paradigm

The realtime communication paradigm has been under the spotlight in recent years due to advances in networking technology and the increasing importance of multimedia communication. Improvements in available bandwidth brought about by the widespread deployment of fiber-optic communication is making it feasible to offer high-speed wide-area communication to larger numbers of users. One of the classes of applications that this technology makes possible is multimedia communication. Initial experiments on the Internet have demonstrated a wide-spread interest in audio-visual communication over computer networks.

Multimedia communication has characteristics and needs different from those of other forms of computer communication. It is important to understand these differences in order to provide acceptable multimedia communication support at a reasonable cost to the client. For one thing, delay and delay variation are performance parameters of higher concern to multimedia traffic, since the acceptability of the service depends on the regular arrival of the data at the computer terminal where it is being displayed. This is especially true if the service being offered is interactive in nature, such as for a video conference. There is a threshold of performance below which the data stream is as good as useless; hence, if the offered load of the network is above the critical level, the network should stop accepting new data streams in order to support the existing streams with acceptable quality of service. Furthermore, some forms of multimedia traffic have a more predictable traffic pattern than computer data. The regular time based nature of video or audio streams makes it possible to efficiently multiplex these streams, while providing hard delay guarantees on the network's performance.

In contrast, packet switched network were traditionally designed to support statistical sharing of the network resources among a large number of computer users. The traditional model of a computer as a data source assumes that data is produced in high rate bursts, with long silent periods between bursts. Markov chain on-off and similar models have been used to analyze the performance of computer networks. The network is designed to run at high average utilization levels, which implies that the sum of the peak data rates of the sources exceeds the capacity of the network, sometimes by several orders of magnitude. This kind of network service (humorously referred to as socialistic service) allows the network to be shared at higher levels of utilization, but also allows congestion to start when more than the average number of sources happen to send their bursts at the same time. The service provided by the network is time variant, in terms of important parameters of network performance such as observed packet delay, or received bit-rate. This has been called the "best effort" service paradigm, or the Available Bit Rate (ABR) service paradigm. The goals of this service contrast sharply with those of the service described previously; here, the attempt is to share the available resources fairly among all the active users.

The current Internet is an amalgamation of packet switched LANs, MANs and WANs with a connectionless internetworking protocol (IP) running on top of the media access protocols to provide routing over the whole internetwork. The Internet is designed for heterogeneity and survivability. However, the service provided is extremely load and time dependent. Sources of variability include congestion, route changes, routing instabilities, and other load on routing machines, such as the processing of route update messages.

Asynchronous Transfer Mode (ATM) networks hold a promise to provide an integrated solution for both traditional computer data and the newer multimedia traffic. At present efforts are under way to make sure that the standards chosen by the ITU (formerly the CCITT) and by the ATM Forum[1] adequately support both kinds of traffic. This is an attempt to isolate and standardize the information about traffic characteristics and performance requirements that the network needs to know in order to efficiently support its

[1] An organization of representatives of industry, network providers and users, working towards the definition of interfaces, services and traffic management for ATM networks.

traffic. At the same time, research into the mechanisms using this information to support these two types of traffic with substantially different characteristics on the same network has been under way for a while.

Realtime communication has been defined (26) as communication with guarantees on performance parameters of interest to the client, such as delay, delay variation and message loss. (30, 32) discusses one approach to the provision of realtime communication guarantees, based on a connection oriented paradigm, using admission control to restrict the realtime load on the network, resource reservation to ensure the availability of sufficient resources to meet the performance requirements, and protective packet/cell scheduling during data forwarding to prevent non-realtime load from disrupting the performance of the realtime connections. This approach has been worked out by the Tenet Group at University of California at Berkeley, under the guidance of Professor Domenico Ferrari.

An implementation of the above scheme is briefly described in Chapter 2. This implementation provides guarantees on network performance which are valid in the absence of network faults. The control mechanisms used in the scheme are based on a *proactive* reservation of the resources and thus no longer hold if the resources are affected by a network failure. To date, the research into realtime communication has been mainly focussed on the fundamental mechanisms to make such a service possible and cost-effective. Thus, not much attention has been paid to advanced facilities such as fault management. This dissertation aims to address this requirement.

In this chapter, we will examine the basic control mechanisms available to us to solve the problem of fault management. We will first identify in Section 1.2 a set of requirements that we would like the mechanisms to satisfy. In Section 1.3, we will classify the mechanisms into two broad categories, *proactive* and *reactive*, and identify the kinds of situations in which each type is useful. We will argue that a single solution is not adequate to deal with faults in realtime networks, and propose a combination of the two approaches to solve the problem. Section 1.4 contains a survey of previous work in survivable network design, fault recovery, and fault tolerance as applied to telecommunications and computer networks, and identify ideas which might be useful in realtime networks. In the process, we will try to point out the differences between the assumptions underlying these networks and those underlying realtime network design, which make the existing work not directly applicable to the present problem. Finally, in Section 1.5 we will define the scope of this work and present an outline of the dissertation.

1.2. General requirements from the control mechanisms

We would like to provide an enhancement of the control mechanisms used for realtime communication, in order to deal gracefully with faults in the network. This should be done without increasing, as far as possible, the overheads or inefficiencies of network utilization when operating in the absence of faults. The applications would like to see their performance requirements met; however, the cost constraint must also be kept in mind. If perfect performance in the presence of network faults is very expensive, in terms of the resources required, then the option of providing a service, where a failure causes a temporary disruption, should be explored, if such a service can provide more efficient use of network resources.

It is important to remember that, in most modern computer networks, failures are a low probability event. A realtime network should be designed to be efficient in the common case. However, in the event of a network fault, the mechanism should react gracefully to continue and provide service to as many clients as possible, and the duration of any service interruption should be small, in order that the service be acceptable to the client. In addition, the timing behavior of the network, if not correctly handled at the lower layers of the network, cannot be corrected by higher layer measures. Thus, the realtime nature of the service, where needed, must exist from the lowest layer up.

In the context of high-speed wide-area networks, the problem of providing fast control mechanisms becomes especially hard. The delay-bandwidth product of fast WANs is large. This means that control mechanisms which require communication with entities at the edge of the network are very slow compared to the transmission time of an individual packet or cell. Hence, the timing characteristics of the control mechanisms become important when choosing one approach over another.

In bandwidth-poor environments, the critical issue shifts from the speed of the recovery to the efficiency of resource usage in the common case. Thus, in such networks, a more slowly reacting approach might be more useful, if it allows more users to enjoy the realtime characteristics of the network. In addition, the requirements of the application, and the ability of the client to afford the price of the network resources, will differ widely. Thus, the network needs to support a range of services, so that, depending on the situation, the correct cost-performance combination may be provided.

Finally, we have to keep the implementation complexity in mind. Such a wide variety of services, if supported from the lowest layer up, might render the network too complicated to build and maintain. We should identify a few powerful mechanisms and provide the interface to allow higher layers to use them to offer the range of services required. Also, rather than designing a separate scheme for each individual network, taking into consideration its topology, delay-bandwidth product, application mix, operating load, and so on, we would like to investigate schemes which work well in a variety of environments. Of course, there may be parts of a particular implementation that can be tuned to optimal performance in a particular environment, but the design should be general.

1.3. Two classes of control mechanisms

Control mechanisms can be categorized into two classes: *proactive mechanisms* and *reactive mechanisms*. These classes serve two very different functions. In the broadest possible generalizations, proactive mechanisms make worst-case provisions to ensure adequate performance in situations that can be predicted. Because of the worst-case nature of the provisions, they may be inefficient, especially if the worst case is unlikely. Reactive mechanisms, however, wait for a situation to arise, before expending the effort of rectifying it. They suffer from a latency of reaction, during which the service provided to the user may be affected to some extent. It also may not be possible to rectify every situation reactively.

1.3.1. Proactive control mechanisms

Proactive network control mechanisms use *a priori* knowledge, for example, of the traffic to be supported, to arrange for resources to be present in the network when they are needed. One example of proactive network control is virtual circuit establishment with bandwidth reservation. In this case the peak rate required to serve the application is known and reserved in advance, guaranteeing good performance during data transmission. The Tenet scheme, to be described in Chapter 2, uses more detailed knowledge about the application traffic in order to share the network more efficiently. Because of its proactive nature, this approach has the property that the performance of the offered service is guaranteed, regardless of the instantaneous load offered to the system by individual applications, provided there are no faults.

Proactive mechanisms are especially well suited to the context of high speed WANs. They minimize control during data transfer, since the rates and control policies are pre-negotiated and each node can act independently of other nodes in the network during data transmission. This makes the network stable, since the fluctuations and instabilities associated with reactive control of a high-speed network from the edge of the network are avoided. It also makes it possible to support non-realtime data on the network and to operate arbitrarily close to the network capacity without violating guarantees for the realtime traffic.

The basic technique for proactive control is to set up a connection in the network, using *a priori* knowledge of the traffic characteristics of the application to ensure that the establishment of the connection will not violate any existing realtime guarantees and the requirements of the current application will also be met. In the Tenet Suite 1, the Realtime Channel Administration Protocol (RCAP) (7, 43) performs the task of proactive control. The scheme and suite are outlined in Chapter 2.

Proactive mechanisms can also be used for applications which cannot tolerate the latency of reaction to an event. For example, if a special application cannot tolerate a temporary disruption of service in the event of a network fault, proactive methods can be used to prevent any disruption of service under limited fault scenarios. This approach is explored in Chapter 6 of this thesis. However, since the proactive action must be taken before the event, the cost is incurred even if the condition which is being protected against never occurs.

In general, proactive mechanisms block some set of resources, in order that these may be used to provide guaranteed service when needed. If the event when the resources will be needed is unpredictable or infrequent, then the proactive solution will necessarily be inefficient. It is possible to mitigate this by allowing other traffic to use these resources on a preemptable basis. However, the more infrequent the event being planned for, the less cost effective the proactive solution, and the question of whether the client is willing to pay the price must be carefully answered. If the time to react to the event is low, or if the client is prepared to put up with the latency of reaction, the reactive approach is preferable. Also, for completely unforeseen events, such as protocol errors, reactive mechanisms must be used.

1.3.2. Reactive control mechanisms

Reactive network control mechanisms perform tasks in response to observed network conditions. Reactive mechanisms have been used to provide flow control (TCP window flow control), congestion control (TCP window shut down), fault recovery (in the cross-connect layer of telecommunication networks), and error control (TCP retransmissions).

Reactive mechanisms take care of conditions which are impossible or inefficient to account for proactively. This approach handles infrequently occurring events efficiently, since, if the condition never occurs, the cost of reacting to it is not incurred. However, this approach involves an unavoidable reaction latency, which becomes more critical as the bandwidth-delay product of the network increases, since the reaction time of the system maps onto a larger number of bits affected by the condition. If the event is infrequent enough, this latency may be tolerable.

Some problems for which reactive solutions are appropriate are listed here.

Fault management. Faults can be dealt with proactively as mentioned in the previous section, but that is an expensive solution, to be used only if the application needs that level of tolerance. For most applications, moving the resource reservations to an unaffected path with sufficient resources would suffice. A temporary disruption of service would be noticed, but after that the guaranteed service would be restored. Many applications would be satisfied with this level of fault protection if the cost were lower than that of a proactive solution.

Route updates. The information on the basis of which routing is performed tracks the changes in resource reservation state as connections are set up and torn down in the network. The only way to do this is to react to network events such as channel establishments and teardowns to update the routing information database.

Garbage collection of resources. If channels are not torn down correctly due to protocol failures or node failures in the network, we need some background audit process to detect these "zombie" reservations and remove them. Otherwise, after a period of network operation, the resources in the network will be steadily depleted, allowing fewer realtime channels to be established.

Error logging and reporting. For maintenance purposes a log of errors and failures in the network needs to be maintained. This can only be done in a reactive manner. The feedback of this mechanism is slow, relative the the speed of automatic fault recovery, since a human is involved in the feedback loop. But it is essential to ensure that the failures in the network are fixed, otherwise accumulated failures, each one hidden from the user by the automatic recovery process, would decrease the capacity of the network.

The high level design of a reactive protocol, the Real Time Control Message Protocol (RTCMP), which deals with fault recovery using the reactive mechanisms developed in Chapters 3 and 4, is described in Chapter 5 of this thesis.

1.3.3. Interactions

The above two classes of network control serve to solve problems in somewhat mutually exclusive areas of fault management. In bandwidth-poor networks, the proactive fault tolerance ideas may not be appropriate, if the reservation of extra resources prevents other users from using the network. Reactive mechanisms, on the other hand, may offer too weak a service in networks with a high delay-bandwidth product, since the latency of reaction is large compared to the packet transmission time. For applications with strict disruption-free service requirements, the cost of the extra resources may be a less important issue. On the other hand, for most applications using interactive multimedia as a personal communication tool, the rare disruptions caused by network failure may not be critical.

Thus, the best network control may be provided by appropriate combinations and interactions between the two classes of mechanisms. Reactive mechanisms may be used to provide recovery for all channels in the network, so that, if a fault occurs, as many of the realtime channels as possible are restored. Proactive mechanisms can be used to provided additional levels of tolerance, when the network capacity permits it and the application requires it.

In addition, we need to provide some proactive support in order to make the reactive mechanisms applicable to realtime networks. For example, the proactive reservation protocol needs to be modified in order to allow the reactive mechanism to move resource reservations in response to network faults. RCAP supports the proactive establishment of realtime connections, but does not support modifications to existing connections. A reactive modification to RCAP, embodied in the Dynamic Channel Management (DCM) protocol (52), is required to support the fault recovery mechanisms. DCM is also described briefly in Chapter 2.

The proactive reservation technique, mentioned in Section 1.3 and described in Chapter 6, protects only against limited fault scenarios, such as single faults or two faults. If a proactive scheme is designed to protect against single failures, then after the occurrence of one failure the connection is vulnerable to a second one. At this time reactive control is needed to restore a level of redundancy so that the proactive protection is restored. This also calls for interaction between the proactive and reactive control mechanisms.

1.4. Previous work

This section describes related work in the areas of design of survivable network topologies, fault recovery, and fault tolerance. The whole of Chapter 2 is dedicated to a description of related work in the area of realtime communication, since much of this information is necessary background towards our objective of enhancing the control mechanisms for fault recovery and fault tolerance.

1.4.1. Survivable network topologies

There exists a body of graph-theoretic work that is useful in characterizing the survivability of a network topology. Some graph-theoretic definitions of network survivability are discussed in (60). A graph is said to be *k-node-connected* if every source and destination is connected by k node-disjoint paths. The *edge-connectivity* of a graph is the equivalent measure if we consider edge-disjoint paths. A *minimal cutset* of a graph is the smallest number of links, the removal of which breaks the graph into two disconnected components. Similarly, an *articulation set* is a set of nodes, the removal of which breaks the graph into two disconnected components. It can be shown that the cardinality of the minimal cutset of a graph is the same as its edge-connectivity, and the cardinality of the minimal articulation set is the same as the node-connectivity (47). The problem of designing minimal-cost networks with a given degree of connectivity is known to be NP-hard, but known heuristics and algorithms with good average case behavior can solve many real-world network design problems (60).

The above models do not take into account the differing failure probabilities of the network components, since they only evaluate the survivability of a network in terms of the number of edge/node disjoint paths. (22) defines a metric of reliability based on the probability that all the host nodes of a network have some path between them. It shows the problem to be NP-complete, but finds an efficient bounding technique. However, probability based models are only useful if information about the failure probabilities of individual components is known, and assumptions such as independence of the failure events are satisfied. They are also specific to particular networks, and not useful for results that are general across many topologies.

The problem with purely topological views of a network is that capacity information is not taken into account. In addition, the functionalities of the various components, such as multiplexors, switches, repeaters, are also hidden. Network designers also need to consider the various protection systems that would reroute traffic in the event of a fault. In this environment, measures of survivability that take into account the amount of traffic surviving some link or node failure have been developed. Such models are described for instance in (14,54). These works use bandwidth as the only measure of the amount of traffic. In the context of realtime networks, the problem is more complex because realtime traffic has other parameters, such as delay bounds, which also need to be considered. In Chapter 4, we will explain why bandwidth based metrics are inadequate, and develop our own metric for the amount of traffic which is based on the resources required to support the traffic in the network. We will use this to develop metrics of survivability that are better suited to realtime networks.

The schemes presented in this dissertation are applicable to network topologies where multiple paths exist between every (source, destination) pair. In fact, the richer the topology, the better the schemes will work. We have not explored the categorization or evaluation of topologies from the point of view of survivability. The topologies on which we run our simulations measure well on criteria of network reliability such as edge-connectivity. We do, however, try topologies with different edge-connectivities to evaluate the sensitivity of our schemes to the topology.

1.4.2. Fault recovery

Survivability has been an important topic of research in the context of telecommunication networks and computer data networks. This interest has been fueled by the commercial importance of providing fault-free service to the users of the telephone network, and the military advantages of fault-tolerant distributed computing and data communications. However, realtime networks are a relatively new concept, and high-level management functionality has just started to be explored. No significant published research in the area of fault recovery for realtime communication has come to this author's attention. However, much of the work in the field of survivability of telecommunication networks addresses similar issues and needs to be considered. Some existing work on the survivability of computer data networks is also described in this section.

1.4.2.1. Telecommunication networks

Fault recovery has been studied in the context of telecommunication transport networks for decades now. An overview of the techniques used in survivability for telecommunication networks was presented in (25) in the framework of a 4-layer model of the network. The layers in the model are the *switched layer*, the *cross-connect layer*, the *multiplex layer*, and the *physical layer*. In any real network, one or more of the layers may be skipped or combined together. From a survivability point of view, especially for telephone networks, the switched layer and the cross-connect layer are important.

Switched layer network: The nodes in this layer are circuit or packet switches. The links are the trunks (e.g. DS1, DS3) provided by the cross-connect layer. Example networks are Public Switched Traffic Networks (PSTNs), Common Channeling Signaling (CCS) and public packet switched networks (Internet). Survivability in the PSTN at the switched layer is obtained through traffic management. The routing tables are updated to ensure that new calls are correctly routed around the failed link. However, calls in progress on the failed elements are lost, and must be redialed by end systems [15,41].

Cross-connect layer network: The nodes at this layer are the network elements with Time Slot Interchange (TSI) cross-connect capability such as Digital Cross-connect Switches (DCSs) or Add Drop Multiplexors (ADMs) with TSI capability. A TSI is a realtime switch which connects one channel to another, but in a less dynamic form than the switches at the switched layer. The cross-connect layer uses DCS restoral methods, changing cross-connects to restore service by routing existing failed demands onto alternate routes. This is different from what happens at the switched layer, where calls are lost. If the cross-connect layer manages to recover from a fault before the switched layer times out its trunk, the calls on the trunk will just notice a glitch. Most DCS restoral methods are targeted for broadband DCS networks due to the smaller number of demands to reroute.

The demands for this level of the network are the trunks which the higher level switched network uses as links. These trunks have fixed bandwidth requirements, and the routes are static (except during reconfiguration). As such, this network is relatively stateful, and the reconfiguration process attempts to preserve this state. This makes the techniques used in this layer of special interest to us, since realtime networks also have per-channel state (although more complex since the number of parameters is larger).

The DCSs detect a fault and either inform a centralized system or instigate a distributed recovery. Centralized methods [12,65,66] use a separate control network to gather the network state at a central computer and then run flow optimization algorithms to compute a quasi-optimal new configuration. The problems with centralized methods include the need for a separate fault-tolerant control network connecting the DCSs to the central computer, the reliability of the central computer, the time to run the flow optimization algorithm, and the communication overhead of acquiring the network state and disseminate the new configurations to the DCSs. The main advantages of centralized methods are the ability to compute quasi-optimal solutions based on global knowledge and the ability to work with non-intelligent DCSs. (12) also claims that DCS recovery methods based on centralized algorithms have lower communication overhead than distributed methods. This is because for the cross-connect layer networks the information about the current configuration is already at the central computer (from which it was down-loaded during the last reconfiguration) and only the information about the fault(s) needs to be sent to the central computer. However, the dissemination of the new configuration from the central computer to the DCSs takes time, and this may make the communication overhead of a centralized method higher than that of a well designed distributed method, where each DCS computes its own configuration information.

Distributed methods rely on some variation of 'flooding' to reach consensus on the new configuration after the occurrence of a fault. The new configuration may be looked up using the failure state as an index into a static table of pre-computed configurations, or the network elements may dynamically compute their configurations using a distributed algorithm. The pre-computation schemes are limited by the space needed to store the configurations for each possible failure state, and by the time to compute reconfigurations for each possible failure state each time the basic configuration changes. Often only a subset of all failure states are covered, and other states are handled sub-optimally using some fall-back heuristics (21).

The dynamic distributed schemes using greedy algorithms with limited information usually compute sub-optimal configurations, but they are more scalable (34,40,56,64). They may be faster than the centralized methods if the communication overhead is kept under control. Since the control messages are sent on the data network, a separate control network is not required. This makes the distributed method cheaper but also more susceptible to loss of control messages. Part of the work may be pre-computed, for example, by pre-computing a number of alternate routes for each (source, destination) pair, and then choosing the cheapest when a fault occurs using some cost function based on the current network state (10). The basic tradeoff here involves the opposite goals of reducing the latency of the computation and the goodness of the solution. The robustness of the algorithm (in the face of multiple failures of the network on which the control information is being exchanged) is also an issue.

The rerouting techniques can also be orthogonally classified as end-to-end or link rerouting (25). Link rerouting (also known as backhauling or local rerouting) replaces the failed link by a path through the network without changing the rest of the route. It has the advantages of speed and ease of implementation, because there is no need to determine the end points of the trunks for link rerouting. However, it is hard to extend to cover multiple failure and node failure cases. End-to-end (or global) rerouting finds a completely new path from the source to the destination node. It uses network capacity more efficiently, is easier to adapt to different grades of service, and handles node and multiple failures better. However, it is more complex and slower to respond.

In the example in Figure 1.1, the connection on route ABCDEF suffers a failure on link CD. The link rerouting technique would attempt to reroute the connection to ABCGHDEF, unless the path is unusable due to insufficient bandwidth. The global method, trying to find the shortest path with sufficient bandwidth, might choose AIJKLMF. The exact path chosen would, of course, depend on the existing reservations on the links.

Return to normal involves rerouting demands back to the restored path without disrupting service. It is important to restore spare capacity as planned and load balance

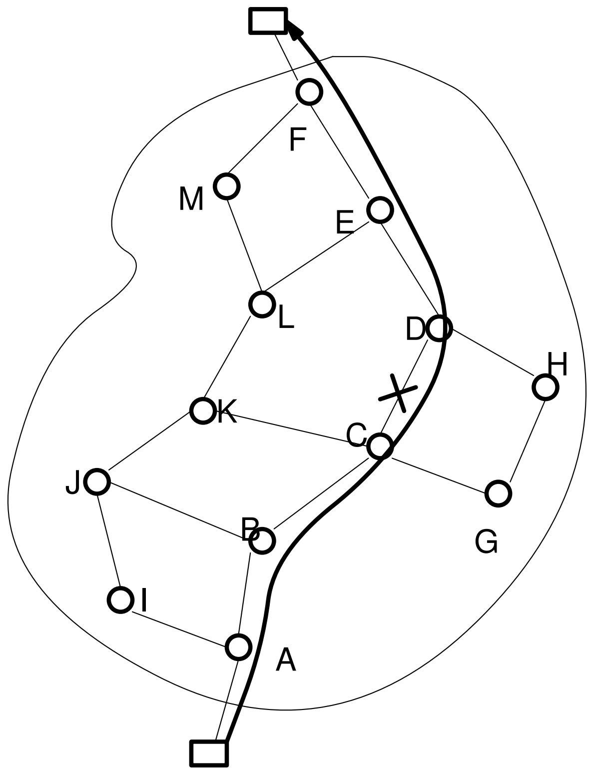

Figure 1.1: Link vs. Global rerouting

the network, since the reroutes are over longer routes and use more resources. However, the time scale of the return to normal is longer, since immediate reaction is not essential, and slower techniques that lead to better final solutions are usually preferred.

Multiplex layer network: The nodes at this layer are terminal or add-drop multiplexors, while the links are high-rate fiber systems. The principle methods at this level are self-healing rings (2, 11) and 1:1 diversely routed Automatic Protection Switching (APS). They use distributed detection and instigation of restoral. Route selection uses simple, precomputed, and static schemes, such as switching from "outer" to "inner" rings in SONET self-healing rings, or switching to a standby fiber in APS. The spare capacity is only used in failure cases, so the switching is local and very fast; return to normal is also simple.

Physical layer network: The links in this case are the fibers. At this layer it is important to ensure the diversity requirement, that is, that paths that should be disjoint in the higher layers should be physically separated.

1.4.2.2. Computer data networks

Conventional computer data networks are mostly based on a connectionless paradigm (e.g., the Internet). Fault recovery in these networks consists of recomputing routing information to route new data correctly. In the Internet no performance guarantees are given, though the protocols attempt to reduce congestion and network instabilities. (46) describes early work on the Internet routing algorithms to correct routing instabilities and improve performance. (67) uses simulation to analyze the dynamics of the routing algorithms with the same objective. However, these algorithms are for connectionless data transmission without any performance guarantees in the network. Hence the survivability techniques do not need to maintain network state for individual connections during reconfiguration. The problems they are attempting to solve are different from those found in survivability for realtime networks.

AN1 (55,57) and AN2 (58) are local area networks designed for high survivability. AN1 is packet switched and connectionless, while AN2 is ATM based, connection-oriented and supports CBR (fixed bandwidth) traffic. CBR traffic is a special case of realtime traffic, hence, some of the problems addressed by AN2 reconfiguration are similar to those faced in realtime networks. However, both networks stop data forwarding during network state acquisition and routing table computation, in order to avoid the problems of cell looping and traffic instabilities. This technique works in local area networks; however, the latencies involved in reconfiguring a wide-area network are sufficient to make the idea of stopping all data forwarding (including unaffected channels) unattractive. This is an example of the difficulties imposed by high-speed WANs on reactive control mechanisms.

1.4.3. Fault tolerance

We define fault-tolerant realtime communication to include only schemes which offer realtime guarantees that are valid in the event of restricted classes of failures. We need to specify the additional clause "restricted classes of failures", because if every component in the network fails simultaneously, there is no technique that can maintain any realtime guarantees. The restrictions imposed on the classes of failure depend on the particular scheme; some examples are single failures, *n*-failures, isolated failures, and so on.

In (73) the concept of Single Failure Immune (SFI) channel is presented. The basic technique used is to start with a basic realtime channel as defined in Tenet scheme, and then augment the channel with additional bypass links to maintain connectivity for each single failure. In the context of the HARTS network (59), this idea has been extended to Isolated Failure Immune (IFI) realtime channels (74). These papers represent the only published approaches to fault tolerance for realtime networks that have come to our attention. They will be described in more detail in Chapter 6.

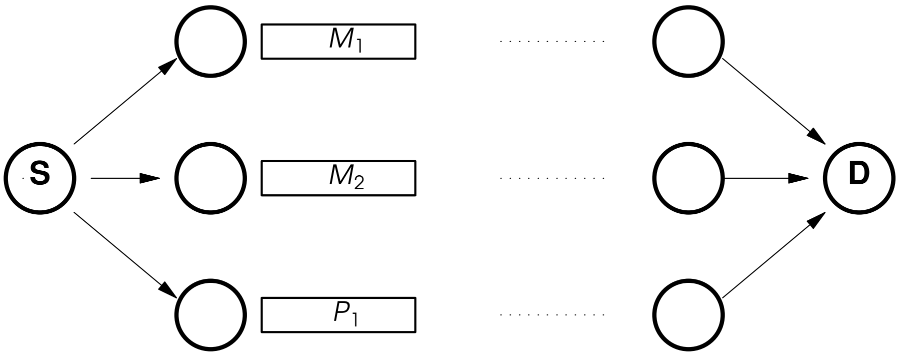

Figure 1.2: Dispersity routing - $(3,2,2)$ **system**

Dispersity routing is the idea of breaking a message into a number of smaller parts, augmenting them by one or more error correcting code or parity messages, and then sending all the messages on disjoint paths through the network to the destination. Different choices on the number of message fragments and the number of parity messages give different dispersity systems. Dispersity routing was first proposed by Maxemchuk in (44), where the schemes were categorized as (N, J, K) systems according to the number of paths in the network (N), the number of message fragments (J), and number of messages needed to be received at the destination to reconstitute the original (K). Dispersity was used to reduce the delay and the variability of message transfers across a store-and-forward network. The analysis was done under the assumptions of Poisson arrivals and steady state. At low network loads, the expected delay of a dispersity system is lower than that of the conventional one-copy scheme. The basic advantages stem from the smaller message size (which reduces the expected transmission delays in the network). By adding redundant messages, the need to wait for the last message is removed, decreasing the delay and delay variation of the system, provided the network is not heavily loaded. A $(3, 2, 2)$ system is shown in Figure 1.3, where an original message is broken into two equal fragments and a parity message constructed for the third path.

Further work by Maxemchuk explored the use of dispersity routing to enhance reliability, share a small set of channels among a number of bursty users with low blocking probability, and handle unexpected traffic sources in the network better.

(17) explored two-copy routing of datagrams in connectionless packet switched networks. Under Poisson arrival and steady-state assumptions, for lightly loaded networks with unreliable links, the expected delay is lower. An algorithm to find the optimal routes for the two-copy scheme is presented in (18), which finds the optimum of a non-linear programming representation of the routing problem using a feasible-descent approach. The algorithm has high but not exponential time complexity.

The computation of the error correcting codes used for the dispersity systems with redundancy is simple for the case where there is only one parity message. In this case, the $(N, N-1, N-1)$ systems, the parity message can be computed as a bit-wise XOR of the other messages. For more than one parity message the class of codes known as Reed-Solomon codes (42) allows us to obtain (N, J, J) systems, that is, provides the ability to decode the message when any J of the N messages have arrived, if J original messages and $N - J$ parity messages were transmitted. Such codes are said to be maximum-distance separable codes. (5) described a maximum-distance separable code which can be implemented in parallel hardware for high-speed communication. The coding scheme belongs to the class of Reed-Solomon codes. (6) extends the coding scheme to the analog domain for implementations at optoelectronic speeds. A software implementation of Reed-Solomon codes can run just fast enough for video conferencing applications on fast workstations (39) (1.2 Mbps on SPARC 10 using 60% of the CPU cycles for coding or decoding). Research is in progress (1) to reduce these computational requirements further, and provide multiple levels of redundancy.

In Chapter 6 we look at the application of some of these ideas to providing fault-tolerant realtime communication.

1.5. Scope of the dissertation

This dissertation investigates techniques and mechanisms to provide fault recovery and fault tolerance for networks that offer realtime services. We investigate a reactive scheme that provides fault recovery by rerouting realtime channels when a failure occurs. We also examine a proactive scheme for fault tolerance, where reservations on multiple paths are used to prevent or limit the degradation of service in the event of a network failure.

The techniques and mechanisms are applicable to a wide class of networks, which can be described by a general realtime network model to be presented in Chapter 3. However, the simulation evaluation and protocol design must be presented in the context of a specific realtime communication environment. Chapter 2 presents a description of the Tenet Realtime Protocol Suite 1, to provide the necessary background information. It also contains a survey of other realtime schemes, from which a set of fundamental principles are extracted to form the network assumptions presented in Chapter 3. These assumptions are applicable to a wide variety of realtime networks, including ATM networks with QoS support. Thereafter, we present a general framework for the recovery process, and then proceed to fill out the details by making a number of design decisions.

The first few decisions, made on the basis of the assumptions about realtime networks, identify the region of the solution space that we will explore. We further narrow the solution space by choosing not to explore a number of issues.

Chapter 4 presents a simulation study of the remaining variables in the fault recovery mechanisms for realtime networks. We measure the performance of these schemes on a number of metrics, chosen on the basis of the requirements specified in Section 1.2. We also explore the effect of varying external factors such as the topology, the network load, the traffic mix, and the failure scenarios on our recovery schemes. As we shall see, the chosen solution space contains a scheme that gives very good performance on all our metrics, and is also tolerant to changes in the external factors. The decisions made in Chapters 3 and 4 are summarized in Chapter 5, and then embodied in the high-level design of a reactive protocol for fault recovery.

Chapter 6 presents one approach to applying proactive reservation techniques to provide fault tolerance for realtime networks, a simulation evaluation of the benefits and costs of this approach, and a comparison with existing solutions. The approach is based on the dispersity systems described in the Section 1.4.3. The evaluation shows that this idea can be used to provide a range of realtime fault-tolerant services, where the level of tolerance may be increased at the expense of increased network capacity requirements, or allowed to degrade gracefully with the number of faults in the network. The network support required is minimal; as long as the network exports the correct set of simple and powerful primitives, and a basic realtime service, the higher layers can combine them appropriately to provide arbitrary levels of tolerance to the user.

Chapter 7 concludes by summarizing the mechanisms presented in this dissertation, evaluating them in the light of the original requirements, and discussing some open issues.